UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Damijan Kuhar

Orodje za razvoj SQL stavkov in delo z relacijskimi podatkovnimi bazami

DIPLOMSKO DELO NA UNIVERZITETNEM ŠTUDIJU
SMER PROGRAMSKA OPREMA

Mentor: prof. dr. Marko Bajec

Ljubljana, 2016

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

V okviru diplomskega dela razvijte aplikacijo za pomoč pri sestavljanju SQL stavkov in poizvedovanjem nad poljubno relacijsko bazo. Podprti naj bodo različni SQL dialekti. Nalogo začnite z analizo zahtev ter načrtom takšne aplikacije. Pri tem preverite tudi sorodna orodja. Za implementacijo uporabite poljubno razvojno okolje.

IZJAVA O AVTORSTVU DIPLOMSKEGA DELA

Spodaj podpisani Damijan Kuhar, z vpisno številko 24940571, sem avtor diplomskega dela z naslovom:

Orodje za razvoj SQL stavkov in delo z relacijskimi podatkovnimi bazami.

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom prof. dr. Marka Bajca;
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela;
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne _____ Podpis avtorja: _____

Zahvaljujem se svojemu mentorju, prof. dr. Marku Bajcu, za vse predloge in nasvete.

Zahvaljujem se tudi družini, prijateljem, sodelavcem in vsem ostalim, ki so me spodbujali pri pisanju te diplomske naloge.

Iskrena hvala.

Pojasnilo:

V pričujočem diplomskem delu so navedeni morda doslej manj znani ali celo neznani izrazi, ki sem jih tvoril med samim delom, tj. razvijanjem omenjenega programa, in sicer je to zahteval delovni proces, saj bi sicer lahko prišlo do neželenih zapletov in nejasnosti.

Poleg tega diplomsko delo vsebuje dosledno zapisane pojme v slovenskem jeziku, ki so del strokovne terminologije, če pa uporaba slednjih ni bila mogoča, sem navedel angleške ustreznice, ki so prav tako ustaljeni termini, ponekod pa sem iz tujejezičnih izrazov skušal po smislu izpeljati slovensko ustreznico.

Imena programov, orodij, izdelkov ipd., ki so v besedilu rabljena pogosteje in niso v slovenskem jeziku, so ob prvi omembi v jedrnem delu diplomskega dela zapisana v narekovajih (npr. "SQL *Plus", "TePro SQL", "coolSQL" …), kasneje pa ne več.

Zavedam se, da zakonitosti slovenskega jezika velevajo, da se samostalniški prilastki nahajajo na desni strani osebka ali predmeta (npr. "stavek SQL" in ne "SQL stavek"), vendar sem zaradi rabe v vsakdanjem življenju ter v izogib nejasnostim tako v naslovu diplomskega dela kot tudi v samem besedilu slednjega tovrstne besedne zveze dosledno zapisoval drugače: SQL stavek, DESCRIBE stavek, SELECT stavek itd.

To pojasnilo naj služi lažjemu branju in boljšemu razumevanju pričujočega diplomskega dela.

Kazalo

Kazalo slik

Povzetek

Program coolSQL, ki ga predstavljam v diplomski nalogi, se je razvil iz začasne forme v enem od projektov, na katerih smo delali v firmi ERPO SISTEMI d.o.o., sčasoma pa je postal kompleksno orodje za lažji in hitrejši razvoj SQL stavkov in posledično tudi aplikacij, ki uporabljajo relacijske podatkovne baze.

Ključne besede

SQL Structured Query Language (strukturirani povpraševalni jezik) – programski jezik za delo s podatkovnimi bazami

DOA, ADO, ODBC
tipi povezav do podatkovne baze

Delphi programski jezik, v katerem je napisana aplikacija

Abstract

The program coolSQL, which is presented in the thesis, has been developed from a temporary form in one of the projects we've been working on in the company ERPO SISTEMI d.o.o., but it eventually became a complex tool for quick and easy development of SQL sentences, and consequently also of applications that are using relational databases.

Keywords

SQL Structured Query Language – programming language for manipulating relational databases

DOA, ADO, ODBC
database connection types

Delphi programming language, which this application is written in

1. Uvod

V firmi ERPO SISTEMI d.o.o. smo pri projektu Terenska prodaja za Pivovarno Laško ali krajše TePro uporabljali podatkovno bazo Oracle, za dostop do baze pa paket komponent DOA – Direct Oracle Access.

Za razvoj SQL stavkov nismo imeli učinkovitega orodja, zato se je firma odločila za ta namen razviti lastno orodje.

1.1 Opis problema

SQL stavke smo vpisovali direktno v ustrezne komponente. Morebitne napake v SQL stavkih so se pokazale šele ob njihovi izvedbi. In bolj ko je TePro postajal kompleksen, dlje časa je trajalo, da se je preklikalo do mesta, ko se je (novi) SQL stavek izvedel. Pogosto je bila napaka samo to, da je manjkal kak zaklepaj, narekovaj, vejica ipd. Razvoj aplikacije je bil zaradi tega počasnejši, kot bi lahko bil.

Drugi problem je bila sama hitrost pisanja SQL stavkov. Vse stavke se je vpisovalo znak po znak, brez pohitritev, brez bližnjic.

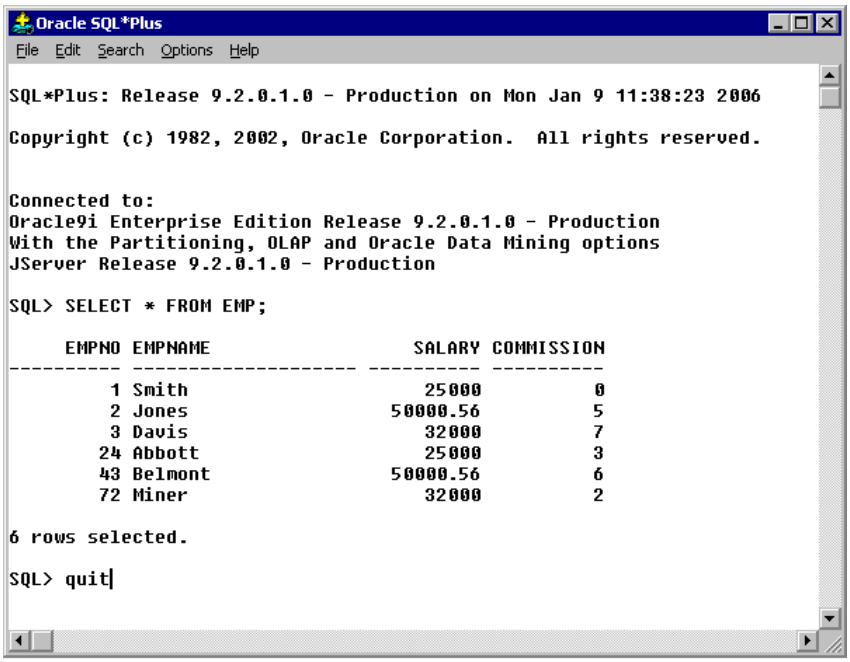
Tretji, zelo pomemben problem, je bila primerjava strukture razvojne baze s strukturami produkcijskih baz.

1.2 Sorodne rešitve

Takrat je bilo na voljo Oraclovo orodje "SQL *Plus". To je bilo tekstovno orodje z ukazno vrstico, in zato precej nerodno za uporabo, predvsem kar se tiče popravljanja in ponovnega izvajanja SQL stavkov. Sicer je omogočalo poganjanje skript – tekstovnih datotek s SQL stavki. Vendar pa je bilo treba te skripte napisati, npr. v Beležnici (Notepad).

SQL *Plus je omogočal prikaz strukture posameznega objekta, npr. tabele, indeksa, procedure ... Rezultat se je prikazal na ekranu. Strukturo vsakega objekta pa je bilo potrebno ročno primerjati. Na ta način je primerjava struktur razvojne in produkcijskih baz trajala nekaj ur.

SQL *Plus je sicer omogočal vse, kar se je potrebovalo za delo z Oraclovimi bazami, vendar je bilo delo z njim zelo zamudno in počasno.



Slika 1: SQL *Plus

1.3 Moja rešitev

Podjetje je imelo željo razviti lastno aplikacijo, ki bi dajala enake rezultate kot SQL *Plus, le da bi bilo do njih možno priti veliko hitreje in enostavneje.

Aplikacijo sem zasnoval tako, da je razširljiva in nadgradljiva.

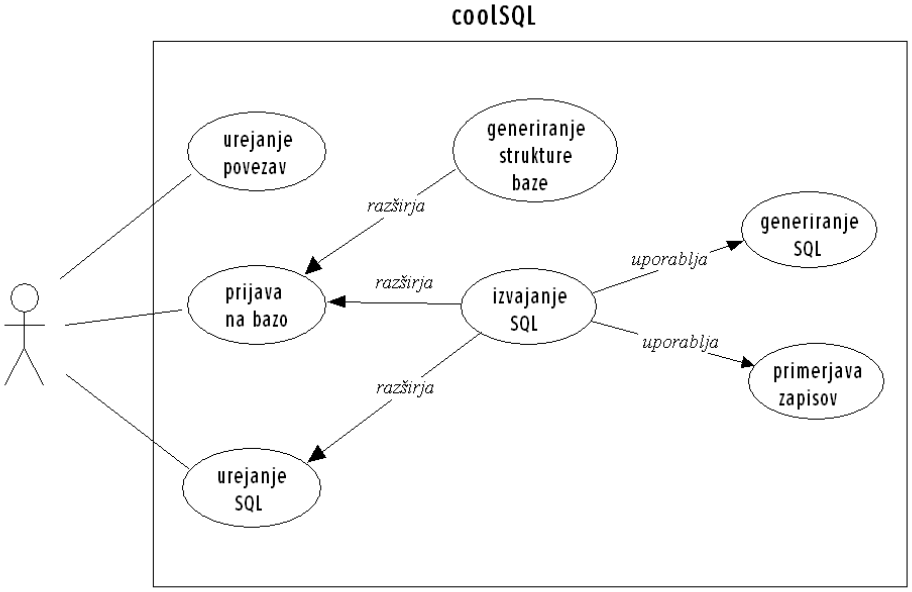
Pisanje SQL stavkov sem pohitril tako, da se za pisanje pogostih ukazov uporabljajo kombinacije tipk. Dodatno so pisanje pohitrile tudi predloge kode (code templates). Imen vseh tabel in polj ni treba vedeti na pamet, niti jih ni treba več ročno vpisovati, ampak se jih izbira iz seznama.

SQL stavke se testira neposredno v urejevalniku in ne šele v aplikaciji.

Preverjanje struktur baz ne poteka ročno po posameznih objektih, ampak se z enim klikom generira tekstovna datoteka z opisom vseh objektov, nato pa se te datoteke primerja z ustreznim programom, npr. Beyond Compare, ki hitro prikaže morebitne razlike.

1.3.1 Analiza in načrt rešitve

Funkcionalne zahteve:



Slika 2: Use case UML diagram

Tok dogodkov za posamezne use case:

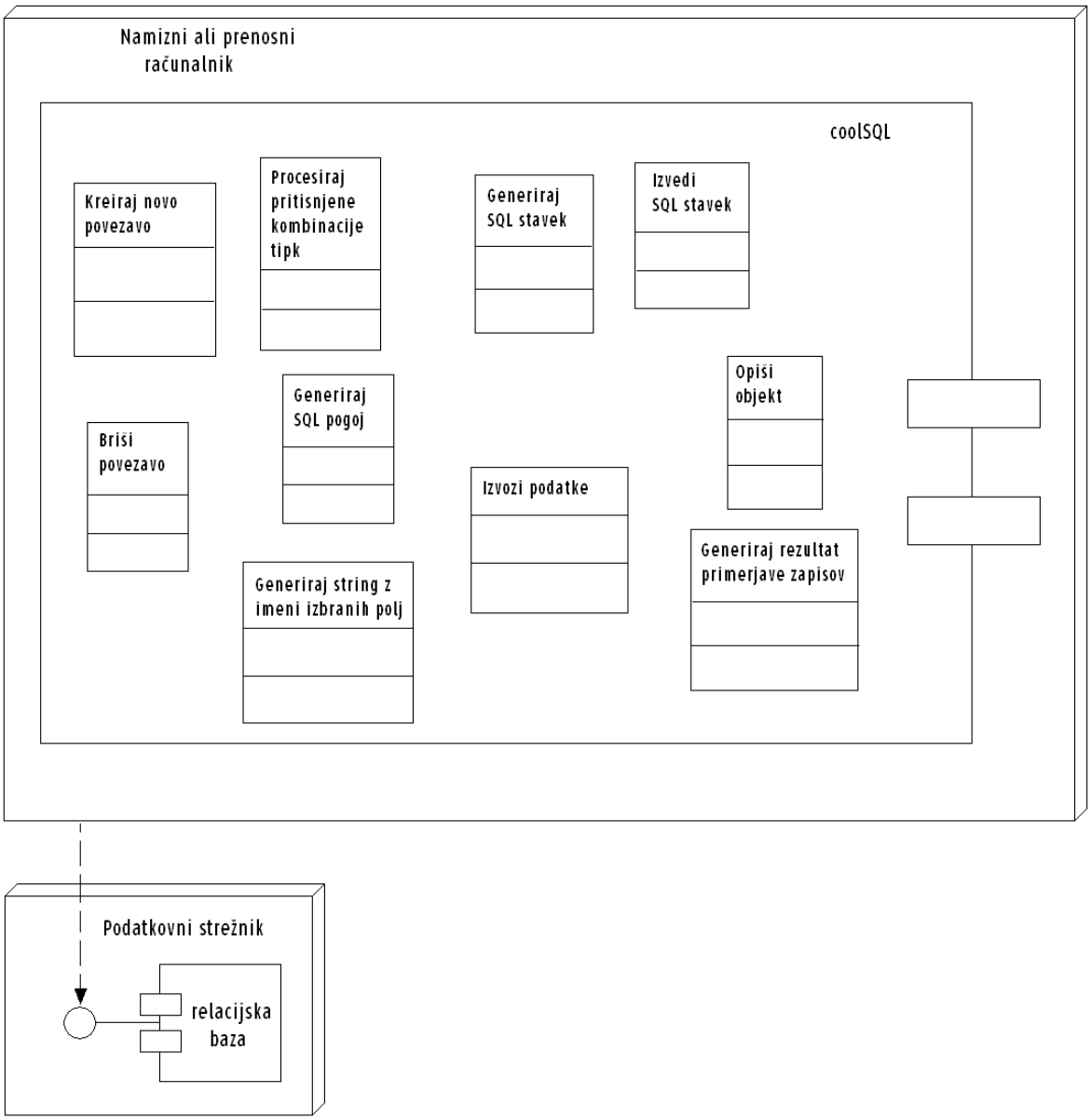
- urejanje povezav:
    - uporabnik doda novo povezavo v seznam povezav;
    - uporabnik pobriše povezavo iz seznama;

    - uporabnik spremeni povezavo (ker se npr. spremeni lokacija baze ali pa geslo za prijavo ipd.);
- prijava na bazo:
    - uporabnik se mora priklopiti na bazo, da lahko izvaja SQL stavke, pregleduje in primerja podatke, preveri strukturo baze ...;
- urejanje SQL stavka:
    - uporabnik lahko vpisuje SQL stavek v urejevalnik, pri tem lahko za hitrejše pisanje uporablja bližnjice s tipkami (keyboard shortcuts);
    - če pa je že povezan na bazo, lahko uporabi tudi izbiranje imen tabel in polj iz seznama;
    - če je povezan na bazo in je že izvedel nek SELECT stavek, lahko uporabi še generiranje SQL stavkov (SELECT, INSERT, DELETE) ali pa samo delov stavkov, npr. pogojev ("polje = vrednost", "polje <> vrednost", "polje like '%%' ", "polje in ( )");
- izvajanje SQL stavka:
    - uporabnik lahko izvede SQL stavek, če je povezan na bazo in ima že napisan SQL stavek;
- generiranje strukture baze:
    - uporabnik lahko generira skripto z opisom vseh ali izbranih objektov, če je že povezan na bazo;
- generiranje SQL stavkov:
    - uporabnik lahko izbere generiranje SQL stavka iz podatkov, ki jih vrne predhodno izvedeni SQL stavek;
- primerjava zapisov:
    - uporabnik lahko primerja od 2 do 5 zapisov, ki jih vrne predhodno izvedeni SQL stavek; izpišejo se seznam imen polj in vrednosti, za katere velja, da se vsaj pri enem od primerjanih zapisov vrednost v polju razlikuje od ostalih.

Nefunkcionalne zahteve:

- aplikacija naj poleg baze Oracle vsaj delno podpira tudi ostale in s tem tudi SQL narečja;
- če je možno, naj se uporabi več niti;
- aplikacija mora biti enostavno razširljiva in nadgradljiva.

Komponentni diagram:



Slika 3: Komponentni diagram

1.4 TePro SQL

V aplikacijo TePro sem dodal "administratorsko orodje", imenovano "SQL okno". To je bila enostavna forma s tekstovnim poljem za urejanje SQL stavka (TMemo), tabelo za prikaz rezultatov (TDBGrid) in gumbom za izvedbo vpisanega SQL stavka.

Hitro se je pokazalo, da bi bilo priročnejše, če bi bilo SQL okno samostojna aplikacija – in tako je nastal "TePro SQL". Bil je vezan na eno bazo – bazo TePro.

Dodana je bila orodna vrstica z nekaj gumbi: gumb za izvedbo vpisanega SQL stavka, gumba za potrditev (commit) in preklic (rollback) sprememb v Oracle bazi, gumb "info" (izpiše se seznam vseh tabel s številom zapisov v vsaki tabeli).

TePro SQL je omogočil testiranje SQL stavkov, še vedno pa je bilo pisanje stavkov relativno počasno, zato so bili v orodno vrstico dodani še gumbi za nekatere takrat najpogosteje uporabljane SQL ukaze, in sicer:

```
select * from |
select count (*) from |
select | from
select count (|) from
delete from |
truncate table |
TO_DATE ('|', 'dd.mm.yyyy')
TO_DATE ('|', 'dd.mm.yyyy hh24:mi:ss')
```

Slika 4: Orodna vrstica v TePro SQL
Zadnja dva ukaza sta za formatiranje kratkega in dolgega datuma v Oraclu.

Vse pogosteje se je dogodilo, da je bilo potrebno razvijati po več SQL stavkov hkrati, zato je bil program popravljen tako, da se je izvedel tisti SQL stavek, ki je bil izbran oz. osvetljen.

1.5 coolSQL

TePro SQL je postajal čedalje uporabnejši, vendar je bil še vedno preveč omejen, da bi se ga dalo uporabljati tudi pri razvoju ostalih projektov.

Odločil sem se, da se v aplikacijo doda možnost povezovati se tudi na druge baze, in ne samo preko DOA, ampak tudi preko ostalih možnih povezav (ADO, ODBC). TePro SQL pa je dobil novo ime – postal je "coolSQL".

Samo ime sem izbral na osnovi imena nekega drugega projekta, imenovanega coolStart, ki je neke vrste "launch pad" za aplikacije, ki so bile razvite v firmi ERPO SISTEMI d.o.o.

coolSQL se je počasi razvijal, kolikor mi je pač dopuščal čas. Nove funkcionalnosti so se dodajale po potrebi. Pri razvoju pa so se pojavljali tudi različni problemi, ki jih je bilo potrebno rešiti.

2. Tehnični problemi

2.1 DESCRIBE stavek za Oraclove objekte

Na začetku razvoja projekta TePro se je baza skoraj dnevno spreminjala in dopolnjevala. Ko smo lansirali novo verzijo, je bilo potrebno ustrezne spremembe narediti tudi na bazah na posameznih distribucijskih centrih Pivovarne Laško. Občasno je bilo potrebno preveriti, če je struktura baze pravilna.

To je takrat omogočalo Oraclovo tekstovno orodje SQL *Plus, vendar pa je bilo treba rezultate ročno pregledovati, da se je ugotovilo, če je struktura pravilna.

Sčasoma sem ugotovil, da je struktura Oraclovih objektov definirana v Oraclovih sistemskih tabelah. Glavna sistemska tabela je ALL_OBJECTS, kjer so našteti vsi objekti z imeni in nekaterimi parametri. Posamezni tipi objektov so definirani v ločenih tabelah, npr. ALL_TABLES, ALL_INDEXES, ALL_FUNCTIONS itd.

Za "dešifriranje" teh podatkov sem uporabil nekakšen "reverse inženiring", in sicer tako, da sem naredil novo testno bazo, kreiral posamezne testne objekte in sproti preverjal spremembe v sistemskih tabelah, te pa primerjal s strukturo, ki jo je izpisal SQL *Plus.

Tako sem ugotovil, kaj kateri podatek pomeni. Testiral sem samo tiste tipe podatkov, ki smo jih uporabljali v projektu TePro, tako da ta funkcionalnost ne zna opisati poljubne Oraclove baze. Če objekt vsebuje polje tipa, ki ga coolSQL "ne podpira", izpiše opozorilo " * * * Nepredviden tip * * * ".

create_function_SUMNIK.sql | create_table_PISO.sql

```
CREATE TABLE PISO (
  ID_PLAN         NUMBER(8)                          NOT NULL,
  ID_STRANKA      VARCHAR2(12)                       NOT NULL,
  TIP             NUMBER(2)                          NULL,
  PISO_TEKST      VARCHAR2(100)                      NULL,
  PISO_USER       VARCHAR2(6)      DEFAULT 'ADMIN'   NULL,
  PISO_SYSDATE    DATE             DEFAULT SYSDATE   NULL)
TABLESPACE TSTEPRO PCTFREE 10 PCTUSED 0
STORAGE (INITIAL 40k NEXT 40k PCTINCREASE 0 MAXEXTENTS UNLIMITED);
```

Slika 5: Opis objekta
CREATE stavek za eno od tabel v TePro bazi (Oracle).

Del kode, ki generira CREATE stavek za Oraclovo tabelo:

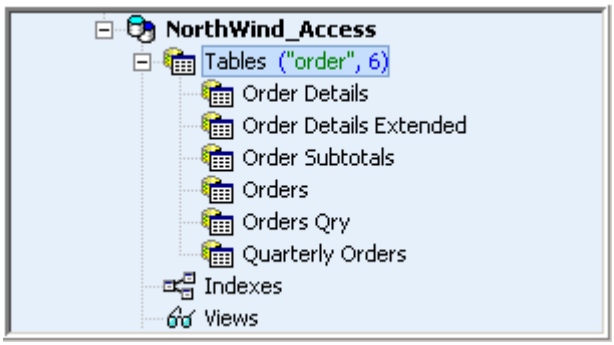
```
  Q.Close;
  with Q.SQL do begin
    Clear;
    Add ('select * from COLS');
    Add ('where TABLE_NAME = ' + #39 + _TableName + #39);
    Add ('order by COLUMN_ID');
  end;
  try
    Q.Open;
  except
    on E: Exception do begin
      StatusBar1.Panels[sbpGeneral].Text :=  '  ' + E.Message;
    end;
  end;

  FillChar (Fields, SizeOf (Fields), 0);
  nofFields := 0;

  maxWidth_ColumnName    := 0;
  maxWidth_ColumnType    := 0;
  maxWidth_ColumnDefault := 0;

  Q.First;
  while not Q.EOF do begin
    nofFields := nofFields + 1;

    Fields [nofFields] .cName := Q.FieldByName ('COLUMN_NAME').AsString;

    txt := '';
    if Q.FieldByName ('DATA_TYPE_OWNER').AsString <> '' then
      txt := txt + Q.FieldByName ('DATA_TYPE_OWNER').AsString + '.';

    txt := txt + Q.FieldByName ('DATA_TYPE').AsString;

    if Q.FieldByName ('DATA_TYPE').AsString = 'NUMBER' then begin
      if Q.FieldByName ('DATA_PRECISION').AsString = '' then
        txt := txt + ''
      else
      if Q.FieldByName ('DATA_SCALE').AsString = '0' then
        txt := txt + '(' + Q.FieldByName ('DATA_PRECISION').AsString + ')'
      else
        txt := txt + '(' + Q.FieldByName ('DATA_PRECISION').AsString + ', ' + Q.FieldByName
('DATA_SCALE').AsString + ')';
    end
    else if (Q.FieldByName ('DATA_TYPE').AsString = 'VARCHAR2') or (Q.FieldByName
('DATA_TYPE').AsString = 'CHAR') then begin
      txt := txt + '(' + Q.FieldByName ('DATA_LENGTH').AsString + ')'
    end
    else if Q.FieldByName ('DATA_TYPE').AsString = 'DATE' then begin
      txt := txt + ''
    end
    else if Q.FieldByName ('DATA_TYPE').AsString = 'LONG RAW' then begin
      txt := txt + ''
    end
    else if Q.FieldByName ('DATA_TYPE').AsString = 'LONG' then begin
      txt := txt + ''
    end
    else begin
      txt := txt + ' ' + ' *** ' + TranslateString (500, 'Nepredviden tip:') + ' *** ';
    end;

    Fields [nofFields] .cType := txt;

    txt := '';
    if Q.FieldByName ('DATA_DEFAULT').AsString <> '' then
      txt := txt + 'DEFAULT ' + Trim (Q.FieldByName ('DATA_DEFAULT').AsString);

    Fields [nofFields] .cDefault := txt;
```

```
    if Q.FieldByName ('NULLABLE').AsString = 'N' then
      Fields [nofFields] .cNull := 'NOT NULL'
    else
      Fields [nofFields] .cNull := 'NULL';

    if Length (Fields [nofFields] .cName) > maxWidth_ColumnName then
      maxWidth_ColumnName := Length (Fields [nofFields] .cName);

    if Length (Fields [nofFields] .cType) > maxWidth_ColumnType then
      maxWidth_ColumnType := Length (Fields [nofFields] .cType);

    if Length (Fields [nofFields] .cDefault) > maxWidth_ColumnDefault then
      maxWidth_ColumnDefault := Length (Fields [nofFields] .cDefault);

    Q.Next;
  end;

  Result := 'CREATE TABLE ' + _TableName + _newLineDescribe_Sequence_True + '(' +
_newLineDescribe_Sequence_True;

  for i := 1 to nofFields do begin
    txt := '';
    txt := txt + '  ' + TrailingSpaces (Fields [i] .cName, maxWidth_ColumnName) + '  ';
    txt := txt + '  ' + TrailingSpaces (Fields [i] .cType, maxWidth_ColumnType) + '  ';
    if (maxWidth_ColumnDefault > 0) then
      txt := txt + '  ' + TrailingSpaces (Fields [i] .cDefault, maxWidth_ColumnDefault) + '  ';
    txt := txt + '  ' + Fields [i] .cNull;
    if (i = nofFields) then
      txt := txt + _newLineDescribe_Sequence_True + ')'
    else
      txt := txt + ',';
    Result := Result + txt + _newLineDescribe_Sequence_True;
  end;

  Q.Close;
  with Q.SQL do begin
    Clear;
    Add ('select * from USER_TABLES');
    Add ('where TABLE_NAME = ' + #39 + _TableName + #39);
  end;
  try
    Q.Open;
  except
    on E: Exception do begin
      StatusBar1.Panels[sbpGeneral].Text :=  '  ' + E.Message;
    end;
  end;

  txt := 'TABLESPACE ' + Q.FieldByName ('TABLESPACE_NAME').AsString +
         ' PCTFREE ' + IntToStr (Q.FieldByName ('PCT_FREE').AsInteger) +
         ' PCTUSED ' + IntToStr (Q.FieldByName ('PCT_USED').AsInteger);

  Result := Result + txt + _newLineDescribe_Sequence_True;

  t1 := IntToStr (Q.FieldByName ('INITIAL_EXTENT').AsInteger div 1024) + 'k';
  t2 := IntToStr (Q.FieldByName ('NEXT_EXTENT').AsInteger div 1024) + 'k';
  t3 := IntToStr (Q.FieldByName ('PCT_INCREASE').AsInteger);

  if cOverrideINITIAL     then t1 := cOverrideINITIALwith;
  if cOverrideNEXT        then t2 := cOverrideNEXTwith;
  if cOverridePCTINCREASE then t3 := cOverridePCTINCREASEwith;

  txt := 'STORAGE (INITIAL ' + t1 + ' NEXT ' + t2 + ' PCTINCREASE ' + t3;

  if (Q.FieldByName ('MAX_EXTENTS').AsInteger = MAXEXTENTS_UNLIMITED) then
    txt := txt + ' MAXEXTENTS UNLIMITED';
  txt := txt + ');';

  Result := Result + txt;
```

2.2 Utripanje (flickering)

coolSQL omogoča vodenje seznama povezav do baz. Povezave so prikazane v drevesni strukturi na levi strani aplikacije. V osnovi so ločene po tipu povezave (ADO, DOA, ODBC).

Ko se povežemo na bazo, se ime (registrirane) povezave odebeli in prikažejo se nova vozlišča, ki predstavljajo tipe objektov v bazi, npr. tabele, indeksi, procedure ... Poleg tipov objektov se v oklepaju izpiše tudi število objektov v bazi, kreira pa se tudi poddrevo z imeni posameznih objektov. Število objektov nekega tipa v podatkovni bazi je lahko veliko, zato sem dodal možnost filtriranja objektov po delu imena – v tem primeru se v oklepaju pred številom objektov izpiše tudi trenutno veljaven filter.



Slika 6: Drevo povezav

Da se poveča preglednost vseh teh podatkov, sem namesto privzetega izrisovanja (črno besedilo na belem ozadju) uporabil način OwnerDraw, kar pomeni, da sem za določene faze izrisovanja moral napisati svoje procedure. Pred začetkom risanja je potrebno pobrisati tisti del Canvasa (preko te lastnosti komponente dostopamo do risalne površine), na katerega se riše, to pa je povzročalo "utripanje" oz. "flickering".

Problem sem rešil tako, da sem uporabil "double buffering", kar pomeni, da se celotno risanje ne izvaja direktno na Canvas, ampak se v ozadju kreira začasna BMP slika ustrezne velikosti, na katero se izrišejo vsi elementi, ki morajo biti prisotni na sliki, nato pa se celotna dokončana BMP slika izriše na ustrezno mesto na Canvasu, ne da bi bilo treba prej na tem mestu pobrisati staro vsebino.

Del kode za izris posameznega vozlišča v drevesu povezav:

```
  NodeRect := kdGetDisplayRect (Node, TV.Canvas, True);

  //draw the selection rect.
  if (cdsSelected in State) then begin
    Brush.Color := $FCDEC5;
    Pen.Color   := $CEA27D;
  end  { if (cdsSelected in State) }
  else begin
    Brush.Color := TV.Color;
    Pen.Color := TV.Color;
  end;
  Rectangle (NodeRect);

  NodeRect := kdGetDisplayRect (Node, TV.Canvas, False);

  offscreenRDB.Width := NodeRect.Right - NodeRect.Left;
  offscreenRDB.Height := NodeRect.Bottom - NodeRect.Top;

  offscreenRDB.Canvas.Brush.Color := TV.Color;
  offscreenRDB.Canvas.Brush.Style := bsSolid;
  offscreenRDB.Canvas.FillRect (Rect (0, 0, offscreenRDB.Width, offscreenRDB.Height));

  offscreenRDB.Canvas.Font.Color := clBlack;
  offscreenRDB.Canvas.Font.Style := TTreeNode_Data (Node.Data^).xFontStyle;

  if (TTreeNode_Data (Node.Data^).xObjectType = otField) then begin
    offscreenRDB.Canvas.Font.Color := DBGridDataTypeTextColor [TTreeNode_Data
(Node.Data^).xFieldType];
  end;

  offscreenRDB.Canvas.TextOut (0, 0 + (NodeRect.Bottom - NodeRect.Top - kdGetTextHeight (TV.Font,
Node.Text)) div 2, Node.Text);

  txt := TTreeNode_Data (Node.Data^).xNodeText_Formated;

  textX := 0;
  while (txt <> '') do begin
    t1 := GetStringToken_byStringDelimeter (txt, '<#');
    t2 := GetStringToken_byStringDelimeter (txt, '#>');

    offscreenRDB.Canvas.TextOut (textX, 0 + (NodeRect.Bottom - NodeRect.Top - kdGetTextHeight
(offscreenRDB.Canvas.Font, t1)) div 2, t1);
    textX := textX + kdGetTextWidth (offscreenRDB.Canvas.Font, t1);

    if (t2 <> '') then begin
      if (Copy (t2, 1, Length ('COLOR=')) = 'COLOR=') then begin
        Delete (t2, 1, Length ('COLOR='));
        offscreenRDB.Canvas.Font.Color := TextToColor (t2);
      end;
    end;
  end;

  offscreenRDB.TransparentColor := offscreenRDB.Canvas.Pixels [0, 0];
  offscreenRDB.Transparent := True;


  TV.Canvas.Draw (NodeRect.Left, NodeRect.Top, offscreenRDB);
```

2.3 Excelove tabele in tekstovne baze

Tudi Excelove datoteke ter CSV in ustrezno sestavljene TXT datoteke je možno interpretirati kot relacijsko podatkovno bazo. Da se poenostavi priklop na tako bazo, sem sprogramiral funkcijo za generiranje povezovalnega niza (ConnectionString), pri čemer je za priklop na bazo potrebno samo izbrati ustrezno XLS datoteko oz. mapo s CSV ali TXT datotekami.

Primer ConnectionStringa za Excel:

```
Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\Temp\test.xls;Extended
Properties=Excel 8.0;Persist Security Info=False
```

Primer ConnectionStringa za tekstovno bazo:

```
Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\Temp\textDB\;Extended
Properties="text;HDR=Yes;FMT=CSVDelimited(;)";Persist Security Info=False
```

V sklopu "Extended Properties" moramo določiti več parametrov, in sicer:

v polje HDR lahko vpišemo:

- **`Yes`** tekstovna baza ima v prvi vrstici napisana imena stolpcev;
- **`No`** v prvi vrstici so že podatki;

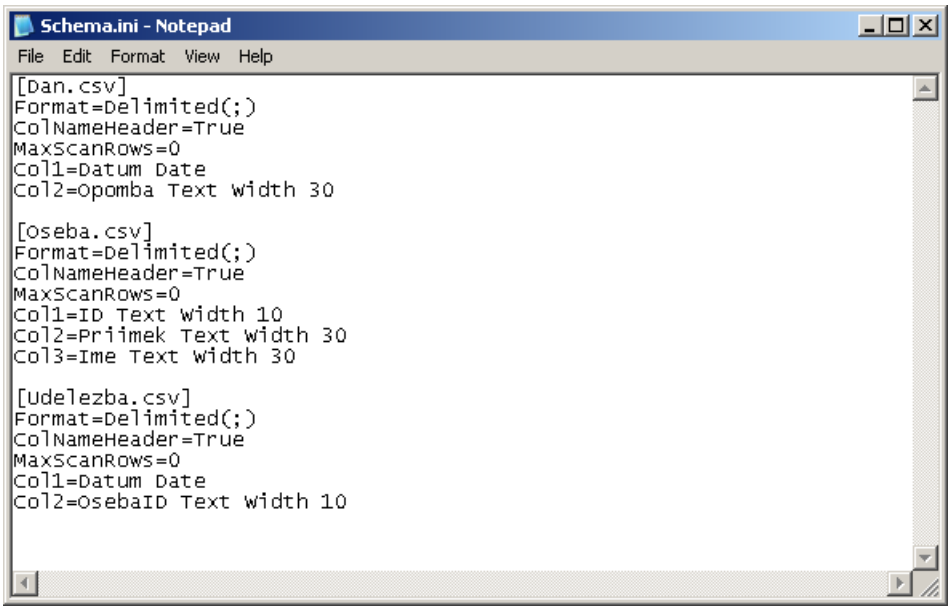
s parametrom FMT pa določimo, na kakšen način so ločeni posamezni stolpci v tekstovni bazi. Izbiramo lahko med:

- **`TabDelimited,`**
- **`Delimited(;),`**
- **`Delimited(,),`**
- **`CSVDelimited.`**

Za CSV oz. TXT bazo velja, da vsaka datoteka predstavlja svojo tabelo. Struktura teh tabel (datotek) pa je definirana v datoteki **`Schema.ini`**.

Struktura datoteke Schema.ini je sledeča:

- vsaka sekcija predstavlja eno tabelo oz. datoteko;
- ime ini sekcije je enako imenu datoteke, na katero se nanaša;
- sledita format datoteke ter podatek o tem, ali so v prvi vrstici imena stolpcev ali podatki;
- nato pa so našteta vsa polja z imeni in tipi.



Slika 7: Primer datoteke Schema.ini

Tipi podatkov so lahko:

- **Bit,**
- **Byte,**
- **Short,**
- **Long,**
- **Currency,**
- **Single,**
- **Double,**
- **DateTime,**
- **Text,**
- **Memo,**
- **Float,**
- **Integer,**
- **LongChar,**
- **Date.**

2.4 Čas izvajanja SQL stavkov

Za odzivnost aplikacije, ki dostopa do podatkovne baze, je zelo pomemben čas izvajanja SQL stavkov. Še posebej je to pomembno, če je število izvajanj veliko, npr. v zanki.

Mnogokrat sem opazil, da na čas izvajanja SQL stavka vpliva marsikaj, npr.:

- v kakšnem vrstnem redu naštejemo tabele, iz katerih črpamo podatke;
- vrstni red pogojev v stavku;
- število zapisov v tabelah.

V coolSQL-u je bilo merjenje časa najprej realizirano z Delphijevo funkcijo za branje trenutnega sistemskega časa (funkcija Now). Merjenje časa je bilo natančno na približno 10 milisekund. Vendar se je v praksi pokazalo, da je to premalo natančno.

Večjo natačnost omogoča API GetTickCount, ki zagotavlja natančnost na približno eno milisekundo.

Največjo natačnost pa je možno doseči z uporabo orodja "High-Resolution Performance Counter", ki omogoča merjenje časa na nekaj 10 nanosekund natančno. Če je visokoločljivi števec prisoten v sistemu, potem API QueryPerformanceFrequency vrne frekvenco, ki pomeni spremembo števca v eni sekundi. V primeru, da števec v sistemu ni prisoten, API vrne vrednost 0.

Koda:

- deklaracija spremenljivk:

```
Var
  TicksStart, TicksEnd     : DWORD;
  { high-resolution performance counter }
  HP_Frequency : Int64;
  HP_Start, HP_Stop : Int64;
```

- tik pred začetkom izvajanja:

```
    TicksStart := GetTickCount;
    QueryPerformanceFrequency (HP_Frequency);
    QueryPerformanceCounter (HP_Start);
```

- takoj po koncu izvajanja:

```
  TicksEnd     := GetTickCount;
  QueryPerformanceCounter (HP_Stop);

  if (HP_Frequency = 0) then begin
    totalMSec  := TicksEnd-TicksStart;
    timeFormat := '%.2d : %.2d : %.3d';
  end
  else begin
    dFrequency := HP_Frequency;
    dStart     := HP_Start;
    dStop      := HP_Stop;
    totalMSec  := 1000.0 * (dStop - dStart) / dFrequency;
    timeFormat := '%.2d : %.2d : %.3d,%.2d';
    if StoparicaVTeku then
      timeFormat := '%.2d : %.2d : %.3d';
```

```
  end;

  if StoparicaVTeku then totalMSec := MainForm.timerStopWatch.Interval * Round (totalMSec /
MainForm.timerStopWatch.Interval);

  dMin     := trunc (totalMSec / (60*1000));
  dSec     := trunc ((totalMSec - 60*1000*dMin) / 1000);
  dMSec    := totalMSec - 1000*dSec - 60*1000*dMin;

  if (HP_Frequency = 0) then begin
    timeText := Format (timeFormat, [dMin, dSec, Round (dMSec)]);
  end
  else begin
    timeText := Format (timeFormat, [dMin, dSec, Trunc (dMSec), Round (100 * Frac (dMSec))]);
  end;

  StatusBar1.Panels [sbpTime].Text := timeText;
```

Slika 8: Primer trajanja izvajanja queryja (11,71 milisekund)
Čas izvajanja queryja je prikazan na desni strani statusne vrstice.

2.5 Širina stolpcev v tabeli rezultatov

Ko SELECT stavek vrne rezultate, se širina stolpcev samodejno nastavi tako, da približno ustreza definirani velikosti polja v strukturi tabele. Pri dolgih tekstovnih poljih se tako pogosto zgodi, da ustrezen stolpec postane celo širši od celotne tabele.

Ker so tako široki stolpci nepraktični, sem implementiral prilagajanje širine stolpcev glede na vsebino v njih. Pri tem pa se upoštevajo še omejitve glede minimalne in maksimalne širine stolpcev, ki se določijo v nastavitvah. Za minimalno širino imamo na izbiro dve možnosti – lahko določimo minimalno širino v pikslih ali pa izberemo, da stolpec ne sme biti ožji, kot je naslov stolpca oz. ime polja. Maksimalno širino pa lahko omejimo tako, da določimo, koliko procentov širine tabele lahko zasede en stolpec.

Mehanizem prilagajanja širine stolpcev je implementiran tudi v Raziskovalcu (File Explorer), kjer se ga sproži s tipkami Ctrl in "+" na numeričnem delu tipkovnice. Tudi v coolSQL-u sem za to uporabil isto kombinacijo tipk.

Koda za prilagajanje širine stolpcev:

```
Procedure TframeDBGrid.AdjustGridColumns;
Const
  widthCellBorder = 6;
Var
  colW      : Array of Integer;
  col       : Integer;
  counter   : Integer;
  w1        : Integer;
  t         : String;
  SavePlace : TBookmark;
  rowDelta  : Integer;
  row       : Integer;
  invisibleColumns     : Integer;
  dataset              : TDataSet;
  progressStep         : Integer;
  progressRefreshCount : Integer;
begin
  TRY
    dataset := nil;

    if not _CoolSQLSettings.DBGrid_CtrlPlus_Functionality then Exit;

    if not Assigned (DBGrid1.DataSource) then Exit;
    if not Assigned (DBGrid1.DataSource.DataSet) then Exit;
    if not DBGrid1.DataSource.DataSet.Active then Exit;

// ------------------------------------------------------------------------
    dataset := THackDBGrid(DBGrid1).DataSource.DataSet;
    rowDelta := -1 + THackDBGrid(DBGrid1).Row;
    row := dataset.RecNo;
// ------------------------------------------------------------------------

    CtrlPlusInProgress := True;

    kdSetCursor (crHourGlass);

    SetLength (colW, dataset.FieldCount+5);
    for col := 0 to dataset.FieldCount-1 do
      case _CoolSQLSettings.DBGrid_MinColWidth_Type of
```

```
        0 : colW [col] := kdGetTextWidth (DBGrid1.TitleFont, dataset.Fields [col].FieldName) +
widthCellBorder;
        1 : colW [col] := _CoolSQLSettings.DBGrid_MinColWidth_Pixels;
      end;

    MainForm.frame_LengthlyOperation11.LengthlyOperation_ShowControls;
    MainForm.frame_LengthlyOperation11.LengthlyOperation_Start (101 {id}, dataset.RecordCount,
progressStep, progressRefreshCount);

    TRY
      counter := 0;
      DS.DataSet := nil;
      SavePlace := dataset.GetBookmark;
      dataset.First;
      while (not dataset.EOF) and not
MainForm.frame_LengthlyOperation11._LengthlyOperationCanceled_ do begin
        MainForm.frame_LengthlyOperation11.LengthlyOperation_Update (dataset.RecNo,
dataset.RecordCount, progressStep, progressRefreshCount);
        for col := 0 to dataset.FieldCount-1 do begin
          t := dataset.Fields [col].DisplayText;

          if dataset.Fields [col].IsNull then
            t := _DBGrid_NULL_Text;
          w1 := kdGetTextWidth (DBGrid1.Font, t) + widthCellBorder;

          if (w1 > colW [col]) then colW [col] := w1;
        end;
        dataset.Next;
        counter := counter + 1;
      end;

      dataset.GotoBookmark (SavePlace);
      dataset.FreeBookmark (SavePlace);
      DS.DataSet := dataset;

      invisibleColumns := 0;

      MainForm.frame_LengthlyOperation11.LengthlyOperation_Start (101 {id}, dataset.FieldCount,
progressStep, progressRefreshCount);

      for col := 0 to dataset.FieldCount-1 do begin
        MainForm.frame_LengthlyOperation11.LengthlyOperation_Update (col, dataset.FieldCount,
progressStep, progressRefreshCount);
        if (_CoolSQLSettings.DBGrid_MaxColWidth_Type = 1) and (colW [col] > (DBGrid1.Width *
_CoolSQLSettings.DBGrid_MaxColWidth_Percent) div 100) then
          colW [col] := (DBGrid1.Width * _CoolSQLSettings.DBGrid_MaxColWidth_Percent) div 100;
        if dataset.Fields [col] .Visible then begin
          DBGrid1.Columns [col+0-invisibleColumns].Width := colW [col];
        end
        else
          invisibleColumns := invisibleColumns + 1;
      end;

    // ---------------------------------------------------------------------
      with dataset do begin
        DisableControls;
        RecNo := row;
        MoveBy (-rowDelta);
        MoveBy (rowDelta);
        EnableControls;
      end;
    // ---------------------------------------------------------------------

    EXCEPT
      on E: Exception do begin
        DisplayError ('NAPAKA AdjustGridColumns: ' + E.ClassName + ': ' + E.Message);
      end;
    END;
  FINALLY
    if Assigned (dataset) and not Assigned (DS.DataSet) then
      DS.DataSet := dataset;
```

```
    kdRestoreCursor;

    CtrlPlusInProgress := False;

    MainForm.frame_LengthlyOperation11.LengthlyOperation_Stop;
    MainForm.frame_LengthlyOperation11.LengthlyOperation_HideControls;
  END;
end;  { Procedure TframeDBGrid.AdjustGridColumns }
```

2.6 Podroben pogled vsebine celic

V tabeli rezultatov nekaterih vrst podatkov ni možno prikazati, recimo dolgih tekstovnih polj (MEMO, WIDEMEMO) ali binarnih objektov (BLOB).

Za premostitev tega problema sem ustvaril *podroben pogled* (detailed view). Ta nam omogoča, da isti podatek lahko gledamo na več načinov. Lahko ga prikažemo kot tekst ali pa ga v HEX editorju pogledamo na nivoju posameznih bytov. Opciji "Web" in "Image" sta bili uporabljani na nekem konkretnem projektu, kjer se je v BLOB polje shranjevala XML koda oz. slika v BMP formatu.

Vsebino BLOB-a pa je možno shraniti tudi v datoteko.

Koda za tekstovni prikaz:

```
  Memo1.Text := MainForm.frameDBGrid1.DS.DataSet.FieldByName
(MainForm.frameDBGrid1.DBGrid1.SelectedField.FieldName).AsString;
```

Koda, ki naloži vsebino v buffer za HEX pogled:

```
  LDataSize.Caption := 'null';

  if MainForm.frameDBGrid1.DS.DataSet.FieldByName
(MainForm.frameDBGrid1.DBGrid1.SelectedField.FieldName).IsNull then Exit;

  if MainForm.frameDBGrid1.DS.DataSet.FieldByName
(MainForm.frameDBGrid1.DBGrid1.SelectedField.FieldName).IsBlob then begin
    blobstream := MainForm.frameDBGrid1.DS.DataSet.CreateBlobStream
(MainForm.frameDBGrid1.DBGrid1.SelectedField, bmRead);
    try
      blobstream.Seek (0, soFromBeginning);
      Size := blobstream.Size;
      if (Size > SizeOf (Buffer)) then begin
        DisplayError ('buffer to small');
        Size := SizeOf (Buffer);
      end;
      blobstream.Read (buffer, Size);
    finally
      blobstream.Free;
    end;
  end
  else begin
    Size := MainForm.frameDBGrid1.DS.DataSet.FieldByName
(MainForm.frameDBGrid1.DBGrid1.SelectedField.FieldName).DataSize;
    if (Size > SizeOf (Buffer)) then begin
```

```
      DisplayError ('buffer to small');
      Size := SizeOf (Buffer);
    end;
    MainForm.frameDBGrid1.DS.DataSet.FieldByName
(MainForm.frameDBGrid1.DBGrid1.SelectedField.FieldName).GetData (@Buffer);
  end;

  LDataSize.Caption := IntToStr (Size) + ' byte(s)';
```

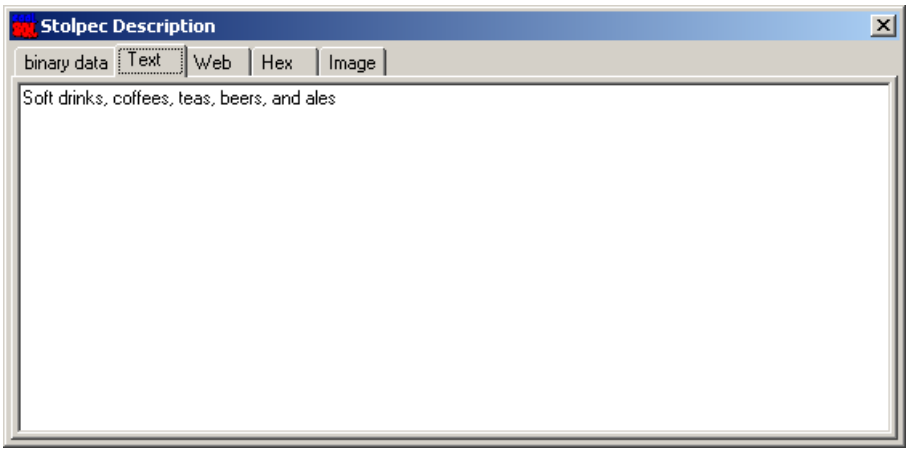
Koda za prikaz kot BMP slika:

```
  if (pageControlDisplayAs.ActivePage = pageImage) then begin
    try
      bmp := TBitmap.Create;
      blobstream := MainForm.frameDBGrid1.DS.DataSet.CreateBlobStream
(MainForm.frameDBGrid1.DBGrid1.SelectedField, bmRead);
      try
        blobstream.Seek (0, soFromBeginning);
        bmp.LoadFromStream (blobstream);
        Image1.Picture.Bitmap := bmp;
      except
        on E: Exception do begin
          MainForm.StatusBar1.Panels [sbpGeneral].Text := EncodeStatusPanelMessageType (mtError)
+ E.ClassName + ': ' + E.Message;
{          bmp.Free;
          bmp := TBitmap.Create;
          bmp.Canvas.Font.Color := clRed;
          bmp.Width := kdGetTextWidth (bmp.Canvas.Font, E.ClassName + ': ' + E.Message) + 8;
          bmp.Height := kdGetTextHeight (bmp.Canvas.Font, E.ClassName + ': ' + E.Message) + 4;
          bmp.Canvas.TextOut (4, 2, E.ClassName + ': ' + E.Message);
          Image1.Picture.Bitmap := bmp; {}
        end;
      end;
    finally
      blobstream.Free;
      bmp.Free;
    end;
  end;
```

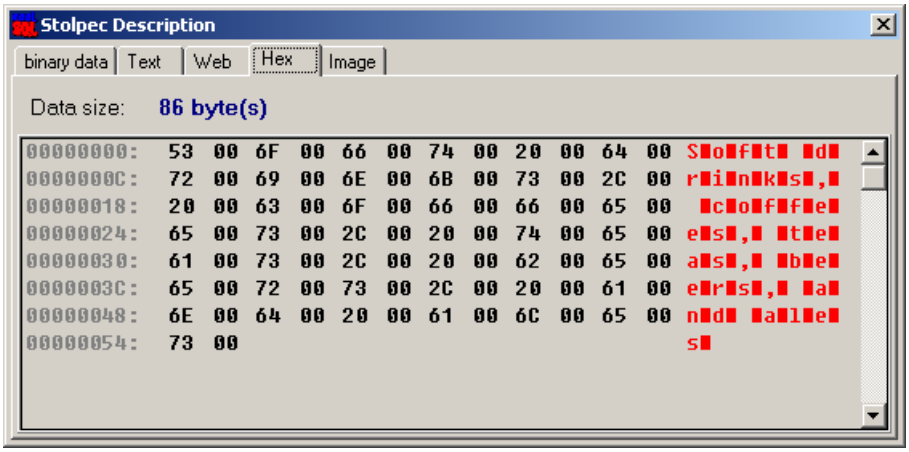
Koda za shranjevanje vsebine polja v datoteko:

```
Function SaveBlobToFile (inField : TField; inFileName : String) : Boolean;
Var
  f          : File;
  blobstream : TStream;
  filestream : TFileStream;
begin
  Result := True;

  if not FileExists (inFileName) then begin
    try
      AssignFile (f, inFileName);
      Rewrite (f);
      CloseFile (f);
    except
      on E: Exception do begin
        Result := False;
        MainForm.StatusBar1.Panels [sbpGeneral].Text := EncodeStatusPanelMessageType (mtError) +
E.ClassName + ': ' + E.Message;
        Exit;
      end;
    end;
  end;
```

```
  try
    filestream := TFileStream.Create (inFileName, fmOpenWrite);
    blobstream := MainForm.frameDBGrid1.DS.DataSet.CreateBlobStream (inField, bmRead);
    try
      blobstream.Seek (0, soFromBeginning);
      filestream.CopyFrom (blobstream, blobstream.Size);
    finally
      blobstream.Free;
      filestream.Free;
    end;
  except
    on E: Exception do begin
      Result := False;
      MainForm.StatusBar1.Panels [sbpGeneral].Text := EncodeStatusPanelMessageType (mtError) +
E.ClassName + ': ' + E.Message;
    end;
  end;
end;  { Procedure SaveBlobToFile }
```



Slika 9: Prikaz MEMO polja kot tekst

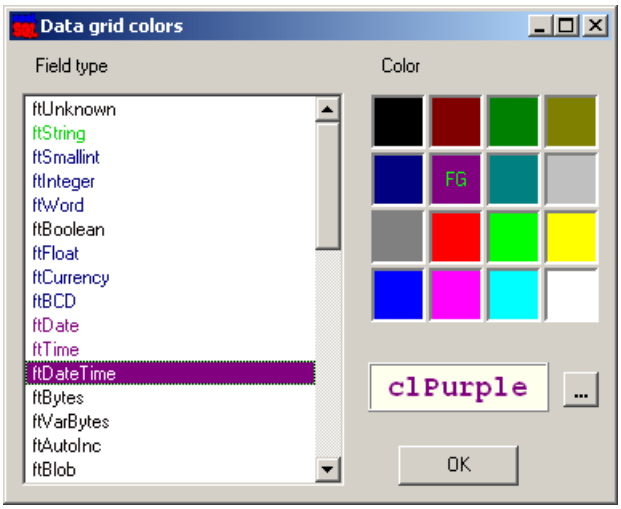


Slika 10: Prikaz teksta v HEX oknu
Iz slike se vidi, da je tekst shranjen v Unicode formatu.

2.7 Tabela za prikaz rezultatov

Privzeto se rezultati izpišejo s črno barvo na beli podlagi ne glede na tip podatka. Da bi bili rezultati preglednejši, sem se odločil, da podatke obarvam glede na njihov tip.

Barve za posamezne tipe podatkov je možno določiti v nastavitvah.



Slika 11: Določanje barv glede na tip podatkov

Koda za izris posamezne celice v tabeli:

```
procedure TframeDBGrid.DBGrid1DrawColumnCell(Sender: TObject;
  const Rect: TRect; DataCol: Integer; Column: TColumn;
  State: TGridDrawState);
Var
  tw, th : Integer;
  tx, ty : Integer;
  t      : String;
begin
  TRY
    if Assigned (extCheckListBox_Fields) then begin
      if (extCheckListBox_Fields.Count > 0) and not extCheckListBox_Fields.Checked
[extCheckListBox_Fields.ItemIndex] then begin
  { - ker se ne da nastaviti DBGrid1.SelectedIndex na -1;
    - v tem primeru se avtomatsko nastavi na 0,
      zato je vedno označeno polje v prvem stolpcu
    - zato nastavimo barve tako, kot da celica ni izbrana
  }
        DBGrid1.Canvas.Brush.Color := clWindow;
        DBGrid1.Canvas.Font.Color  := clWindowText;
        if (DBGrid1.Columns.Count > 0) then
          if Assigned (DBGrid1.Columns [0] .Field) then
            DBGrid1.Canvas.Font.Color  := DBGridDataTypeTextColor [DBGrid1.Columns [0]
.Field.DataType];
      end;
    end;  { if Assigned (extCheckListBox_Fields) }

    if not (gdSelected in State) then
```

```
      if Assigned (Column.Field) then begin
        DBGrid1.Canvas.Font.Color  := DBGridDataTypeTextColor [Column.Field.DataType];
      end;

    if (THackDBGrid(DBGrid1).DataLink.ActiveRecord + 1 = THackDBGrid(DBGrid1).Row) and not
((gdFocused in State) or (gdSelected in State)) then
      DBGrid1.Canvas.Brush.Color := MixColors (DBGrid1.Canvas.Brush.Color, clYellow, 0.8);


    if Assigned (Column.Field) then begin
      if not (gdSelected in State) and (Column.Field.Tag = -1) then begin
        DBGrid1.Canvas.Brush.Color := clRed;
        DBGrid1.Canvas.Font.Color  := clWhite;
      end;
    end;


    DBGrid1.DefaultDrawColumnCell (Rect, DataCol, Column, State);

    if not Assigned (Column.Field) then Exit;


    if Column.Field.IsNull then begin
      DBGrid1.Canvas.Font.Color  := _DBGrid_NULL_Color;
      tw := DBGrid1.Canvas.TextWidth (_DBGrid_NULL_Text);
      th := DBGrid1.Canvas.TextHeight (_DBGrid_NULL_Text);
      case Column.Alignment of
        taLeftJustify  : tx := Rect.Left + 2;
        taRightJustify : tx := Rect.Right - tw - 3;
        taCenter       : tx := (Rect.Right + Rect.Left - tw) div 2;
      end;
      ty := (Rect.Bottom + Rect.Top - th) div 2;
      DBGrid1.Canvas.TextRect (Rect, tx, ty, _DBGrid_NULL_Text);
    end;
  EXCEPT
    on E: Exception do begin
      DisplayError ('NAPAKA DBGrid1DrawColumnCell: ' + E.ClassName + ': ' + E.Message);
    end;
  END;
end;  { procedure TframeDBGrid.DBGrid1DrawColumnCell }
```

2.8 Format zapisa datumov

Iz izkušenj lahko rečem, da je največ razlik med posameznimi SQL narečji v formatu zapisa datuma. V coolSQL-u je to pomembno vprašanje, ker je pri generiranju pogojev (ali pa kar celih SQL stavkov) iz podatkov v tabeli rezultatov potrebno datume napisati v formatu, ki ga podpira baza, na katero smo priključeni.

Primeri zapisov datuma in časa za nekatera SQL narečja:

- splošen format:

```
'dd.mm.yyyy hh:nn:ss'
```

- Oracle:

```
TO_DATE ('|', 'dd.mm.yyyy hh24:mi:ss')
```

- MS SQL Server, DB2:

```
'yyyy-mm-dd hh:nn:ss'
```

- MS Access:

```
#dd/mm/yyyy#
#YYYY-MM-DD hh:mm:ss#
```

- tekstovne baze (CSV, TXT):

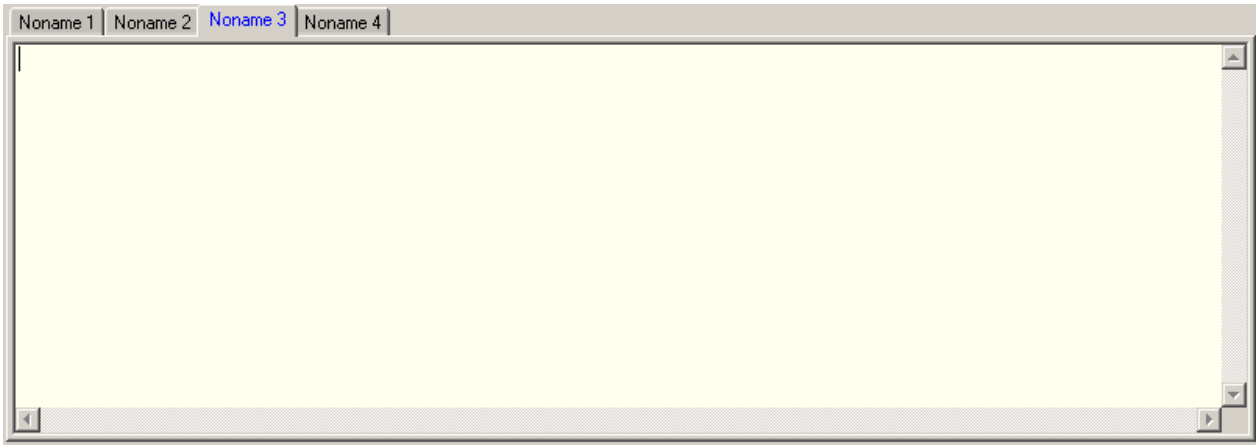
```
#yyyy-mm-dd#
```

3. Predstavitev ključnih funkcionalnosti

Kazalo

3.1 Datoteke

Zelo nujna je bila možnost shranjevanja vsebine urejevalnika v datoteko (ter tudi branje iz datoteke), nepogrešljiv pa je postal tudi seznam 12 nazadnje uporabljenih datotek (t. i. MRU oz. "most recently used" list). Nadvse uporabni so tudi kasneje dodani zavihki (TPageControl), ki so omogočili, da je bilo lahko odprtih več datotek hkrati.

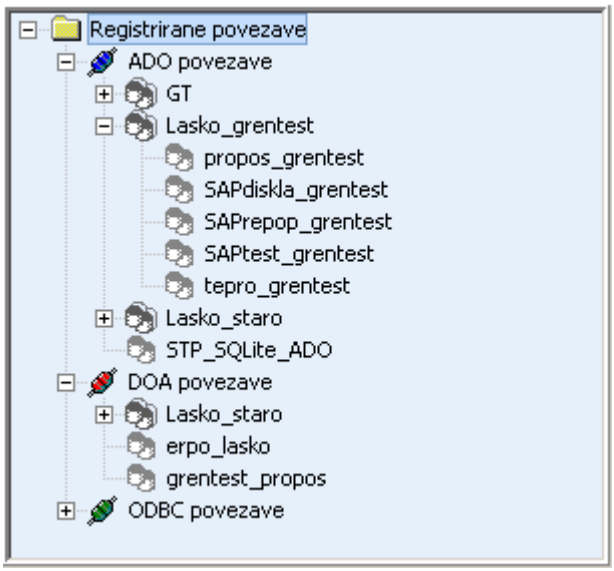


Slika A 1: Zavihki

Slika A 2: Razširjena orodna vrstica
V orodno vrstico so bili dodani gumbi za nov dokument, branje in shranjevanje.

3.2 Povezave do baz

Sprogramiran je bil mehanizem za kreiranje, urejanje in shranjevanje povezav najprej samo do Oraclovih, nato pa še ostalih baz. Povezave so razdeljene glede na tip povezave (DOA, ADO, ODBC). Da se v množici povezav poveča preglednost, je možno ustvariti tudi skupine (sorodnih) povezav.



Slika A 3: Seznam povezav ter skupine povezav

V orodno vrstico je bilo dodanih nekaj gumbov:

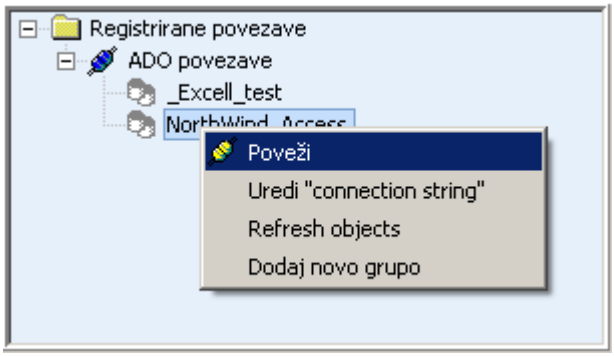
- gumba za dodajanje in brisanje povezav do baz,
- gumba za povezovanje in odklapljanje od baze,
- gumbi za kreiranje začasnih povezav do baz (te povezave se ne shranijo).

Slika A 4: Orodna vrstica za urejanje povezav

Povezave so prikazane v drevesni strukturi na levi strani aplikacije, če pa potrebujemo več prostora za urejanje SQL stavkov, pa lahko drevo povezav tudi skrijemo (menu View >> Databases).

3.2.1 Priklop na bazo

Na bazo, za katero imamo registrirano povezavo, se lahko povežemo na več načinov:
- z gumbom za priključitev na bazo,
- z dvojnim klikom na registrirano povezavo,
- s pritiskom hkrati na Ctrl in Right.

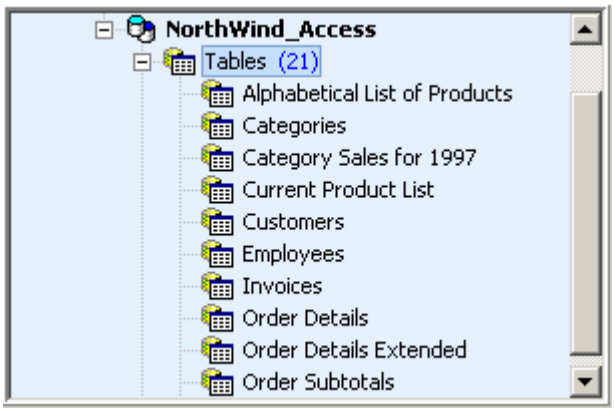


Slika A 5: Povezovanje na bazo

Za odklop od baze lahko uporabimo gumb za odklop, dvojni klik na povezavo ali s tipkami Ctrl in Left.

Ko se povežemo na bazo, se ime (registrirane) povezave odebeli in prikažejo se nova vozlišča, ki predstavljajo tipe objektov v bazi, npr. tabele, indeksi, procedure ...

Slika A 6: Tipi objektov
Poleg tipov objektov se v oklepaju izpiše tudi število objektov v bazi.



Slika A 7: Seznam objektov
Za vsak tip objektov se kreira poddrevo z imeni posameznih objektov.

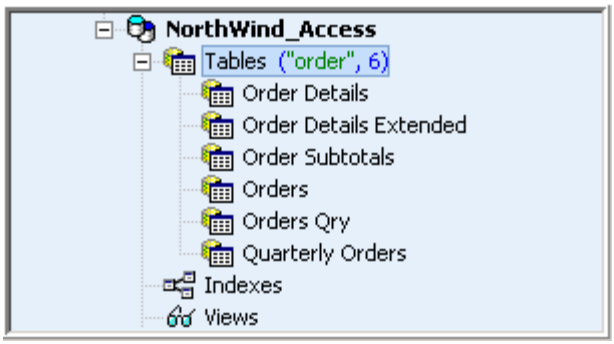
OPOMBA: Ni nujno, da ta funkcionalnost (že) deluje za vse tipe objektov za vse vrste relacijskih podatkovnih baz, kajti v osnovi je bilo to narejeno samo za Oracle, nato pa se je po potrebi dopolnjevalo tudi za nekatere druge vrste baz.

3.2.2 Filtriranje objektov

Število objektov nekega tipa v podatkovni bazi je lahko zares veliko, npr. nekaj sto ali celo več kot tisoč tabel. V takih primerih je zelo uporabna funkcija filtriranja objektov po delu imena.

Objekte filtriramo tako, da izberemo vozlišče, ki predstavlja določen tip objektov. Nato izberemo filtriranje v popup menuju (z desno tipko miške) ali s Ctrl+F. Ko potrdimo želeni filter, se v oklepaju pred številom objektov izpiše tudi trenutno veljaven filter. Filtriranje izklopimo tako, da v filter vpišemo prazen string ali pa v popup menuju izberemo "Briši filter".

Filtriranje lahko prikličemo tudi na vozliščih, ki predstavljajo posamezne objekte določenega tipa.

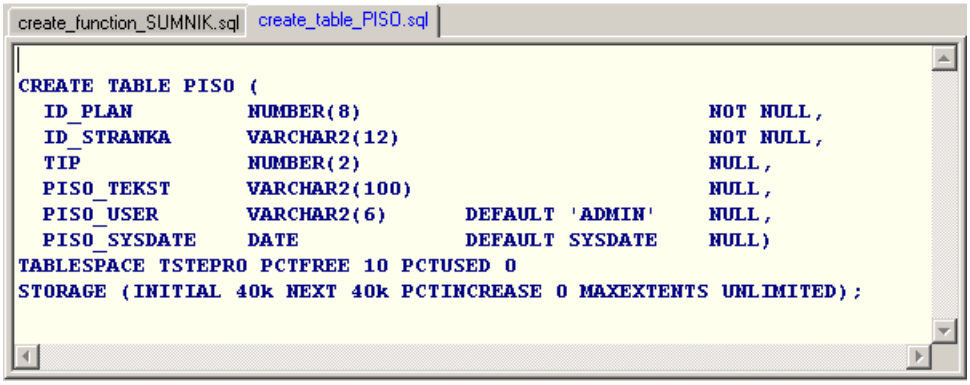


Slika A 8: Filtriranje objektov
Primer: izmed vseh tabel želimo videti samo tiste, ki v imenu vsebujejo "order".

3.2.3 Objekti

Ko urejamo SQL stavek, ni potrebno ročno pisati imena posameznega objekta (tabela, view ...). Z dvojnim klikom na vozlišče, ki predstavlja objekt, ali pa če v popup menuju izberemo ukaz "Izberi", se ime izbranega objekta izpiše v urejevalniku.

V popup menuju sta še 2 ukaza: "Filtriraj" (to je opisano v prejšnji točki) in "Opiši". Ukaz "Opiši" (ali "describe") prikaže, kako je določen objekt v bazi definiran, npr. stavek CREATE TABLE za tabelo ipd. Sam postopek je dejansko neke vrste "reverse inženiring" iz podatkov iz sistemskih tabel. Prav tako je bila tudi ta funkcionalnost v osnovi narejena samo za Oracle, delno pa tudi za nekatere druge vrste baz.



Slika A 9: Opis objekta
CREATE stavek za eno od tabel v TePro bazi (Oracle).

3.3 Urejevalnik

Bolj ali manj običajen tekstovni urejevalnik ima nekaj dodatnih funkcij, ki pohitrijo pisanje SQL stavkov. Nekaterim kombinacijam tipk so prirejeni pogosto uporabljani SQL ukazi, in sicer:

```
Ctrl+S     select * from |
Ctrl+W     where |
Ctrl+O     order by |
Ctrl+G     group by |
Ctrl+I     insert into |
Ctrl+R     rename | to
Ctrl+U     update | set
```

Nekatere kombinacije tipk pa izvedejo določene akcije, in sicer:

`Ctrl+C, Ctrl+V, Ctrl+X` standardne akcije "kopiraj", "prilepi" in "izreži" ("copy", "paste" in "cut"),

`Ctrl+A` izberi vse,

`Ctrl+B` uravnoteženo izbiranje (balanced select),

`Ctrl+E` izvedi izbrani SQL stavek,

`Ctrl+F` iskanje po besedilu,

`Ctrl+J` "code insight",

`Ctrl+N` nov zavihek.

3.3.1 Code Insight – predloge kode

Funkcionalnost "Code Insight" sem povzel po IDE-u (Integrated Development Environment) za Delphi 6, v katerem je coolSQL tudi napisan.

"Code Insight" predloge so shranjene v datoteki coolSQL.CodeTemplates.txt. Vsaka predloga ima v oglatih oklepajih najprej napisano kratico in opis ter opcijsko še bližnjico s tipkami, v naslednjih vrsticah pa je besedilo, ki v urejevalniku nadomesti vpisano kratico.

Predloge je možno definirati v samem coolSQL-u, in sicer v menuju "Tools" izberemo "Code Insight", lahko pa jih, če je coolSQL ugasnjen, popravljamo direktno v tekstovni datoteki coolSQL.CodeTemplates.txt.

Ko v urejevalnik vpišemo izbrano kratico, nato s kombinacijo tipk Ctrl+J sprožimo zamenjavo kratice z definiranim tekstom.

Nekaj primerov iz coolSQL.CodeTemplates.txt datoteke:

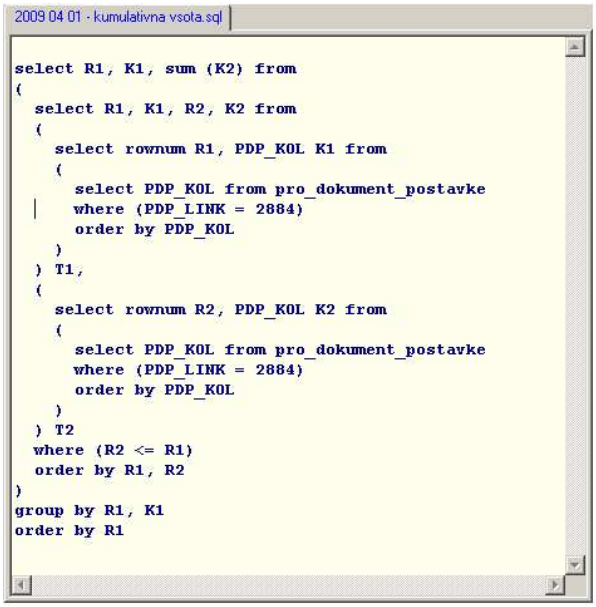
```
[s|select|Ctrl+S]
select * from |

[sd|select distinct]
select distinct | from

[sc|select count]
select count(*) from |

[sm|select max]
select max (|) from

[g|group by|Ctrl+G]
group by |

[o|order by|Ctrl+O]
order by |

[u|update|Ctrl+U]
update |
set

[i|insert|Ctrl+I]
insert into |

[c]
count (*|)

[h]
having |

[hc]
having count (*|)

[d]
distinct |

[ci]
create index |

[cui]
create unique index |

[ct]
create table |

[cts]
create table | as
select * from

[dt]
drop table |

[tt]
truncate table |

[at]
alter table |

[ata]
alter table | add ()

[atm]
alter table | modify ()
```

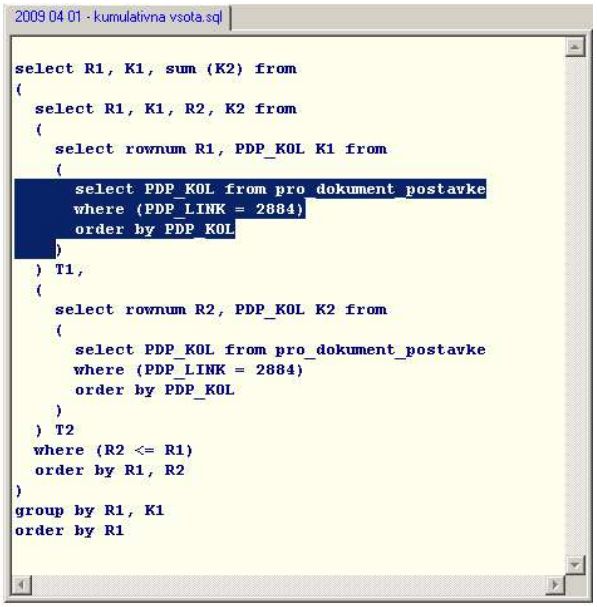
```
[atd]
alter table | drop ()

[r|rename|Ctrl+R]
rename | to

[w|where|Ctrl+W]
where |

[td|TO_DATE ('', 'dd.mm.yyyy')]
TO_DATE ('|', 'dd.mm.yyyy')

[tdt|TO_DATE ('', 'dd.mm.yyyy hh24:mi:ss')]
TO_DATE ('|', 'dd.mm.yyyy hh24:mi:ss')

[tn|to_number]
TO_NUMBER (|)

[dfn|drop function]
drop function |

[df|delete from]
delete from |

[lj|left join on]
left join | on

[in|is null]
is null|

[inn|is not null]
is not null|
```

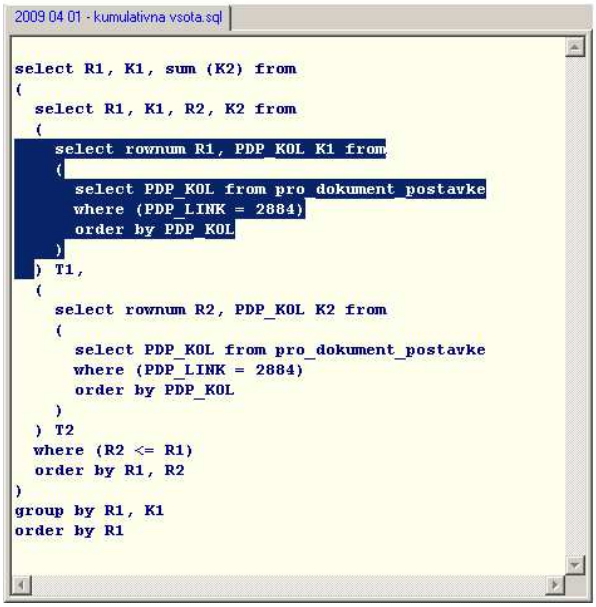
3.3.2 Uravnoteženo izbiranje (balanced select)

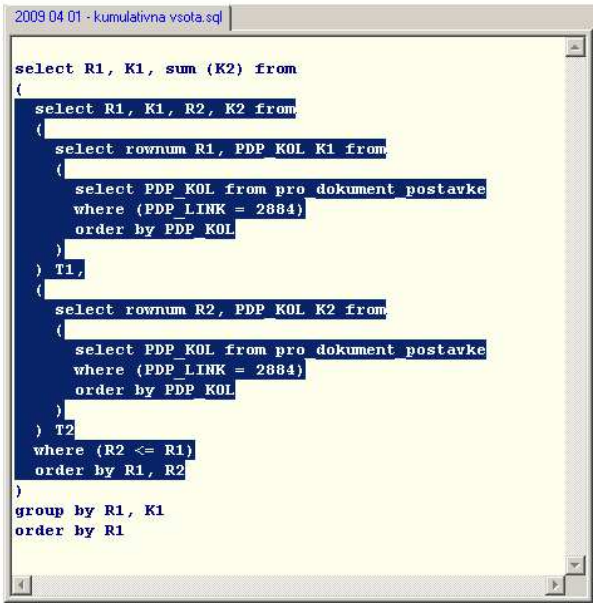
SQL stavki oz. queryji so postajali vedno kompleksnejši z večimi podqueryji, pogosto pa so tudi podqueryji sami vsebovali podqueryje. Da podqueryjev ni bilo treba testirati ločeno, sem dodal t. i. "balanced select", ki deluje po principu veljavnih oklepajnih izrazov. Ob predpostavki, da je kurzor znotraj nekega podqueryja oz. veljavnega oklepajnega izraza, se poišče najmanjši veljavni oklepajni izraz, ki vsebuje trenutno pozicijo kurzorja. Če pa je že (uravnoteženo) izbran nek podquery, potem pa se poišče oklepaj in zaklepaj neposredno "nadrejenega" veljavnega oklepajnega izraza in se izbere celotna vsebina znotraj tega oklepajnega izraza.

V primeru, da se je urejalo več SQL stavkov hkrati, se je pri uravnoteženem izboru zgodilo, da je bila izbrana celotna vsebina urejevalnika, zato sem dodal ločilo med posameznimi queryji, in sicer znak ¤, ki se ga vpiše s Ctrl+Enter (kot Page Break v MS Wordu).









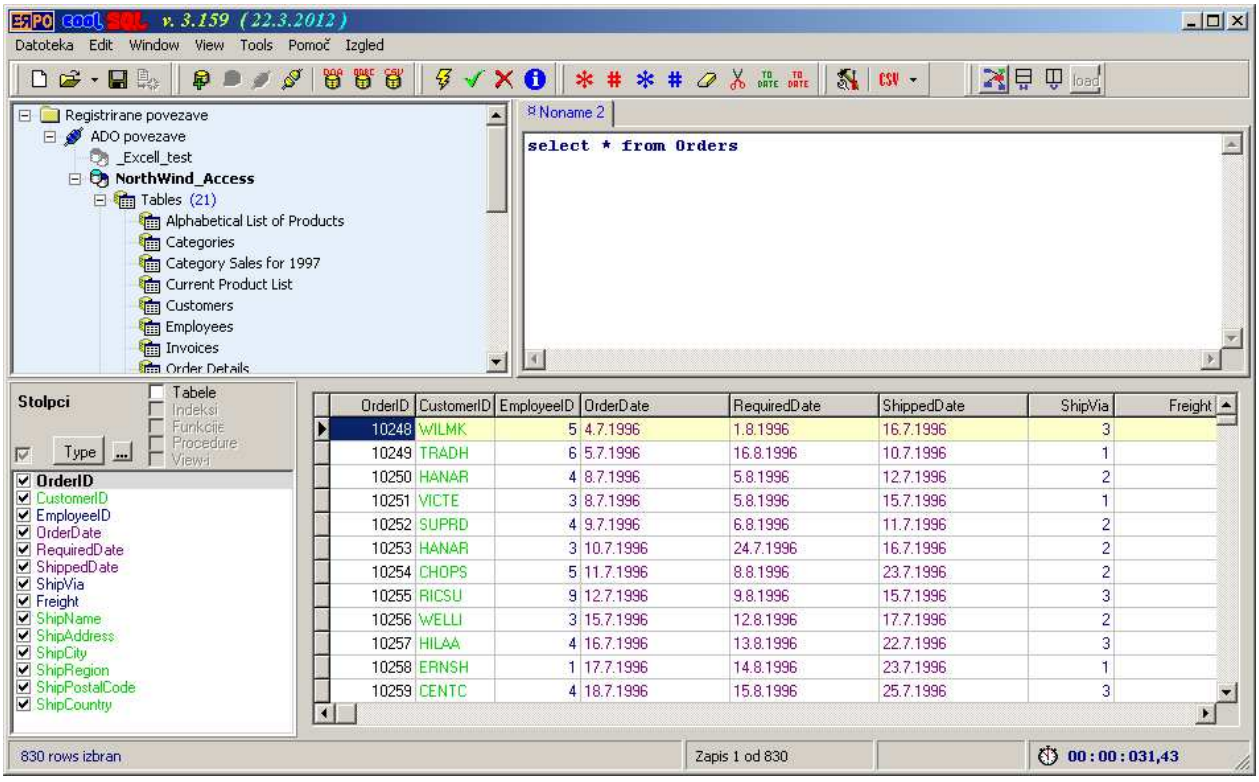
Slika A 10: Uravnoteženo izbiranje (balanced select)

3.4 Tabela z rezultati (DBGrid)

Ko se izvede SELECT stavek, se rezultati prikažejo v tabelarični obliki v spodnjem delu aplikacije. Levo od tabele pa se prikaže seznam vseh prikazanih polj.

Tako imena polj v seznamu kot podatki v tabeli so barvno kodirani glede na tip podatka.

Celice, v katerih je vrednost "null", niso prazne, ampak se z rdečo barvo izpiše "null", in se tako ločijo od celic, ki kot vrednost vsebujejo prazen string.

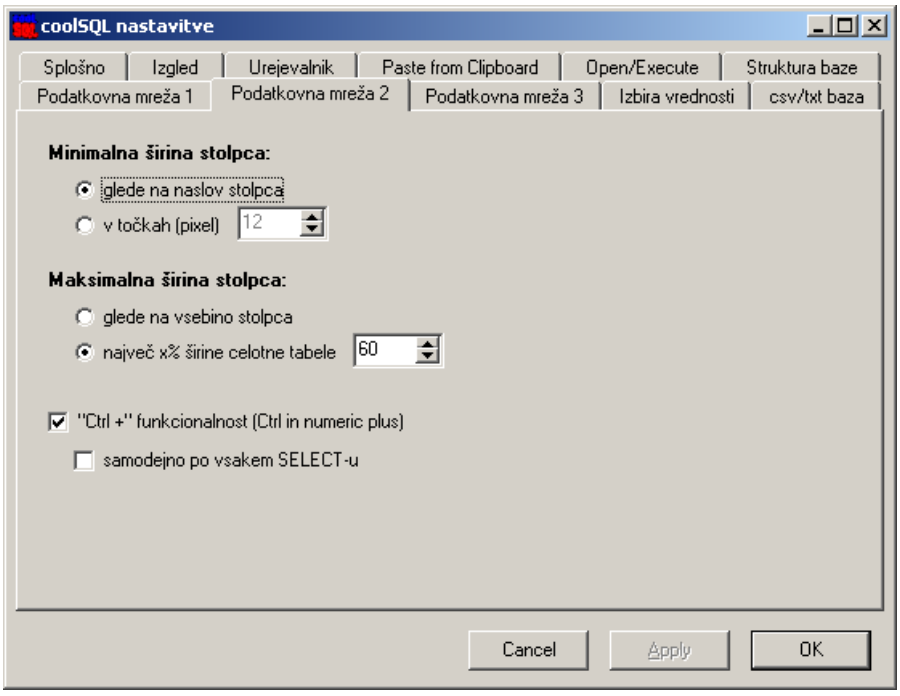


Slika A 11: Seznam polj in tabela z vrnjenimi podatki

3.4.1 Tabela s podatki in seznam polj

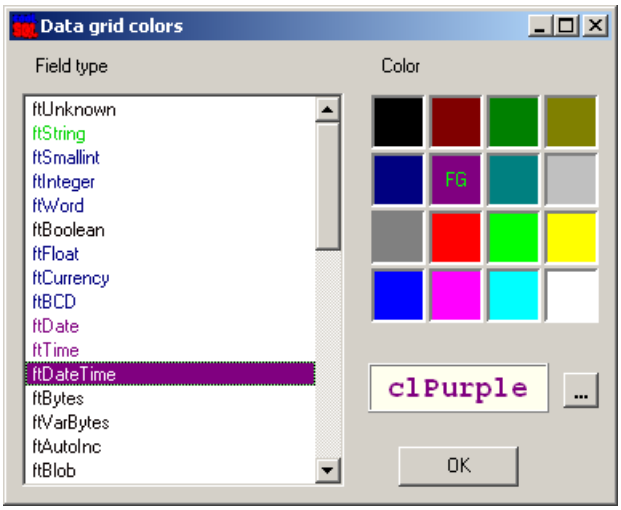
Privzeta širina posameznega stolpca v tabeli je približno sorazmerna z definirano velikostjo polja v strukturi tabele. Da se poveča preglednost podatkov, lahko pritisnemo Ctrl in "+" na numeričnem delu tipkovnice. Širina stolpcev se ustrezno prilagodi podatkom, pri tem pa se upoštevajo še nastavitve minimalne in maksimalne širine stolpca. Te lahko nastavimo, če v orodni vrstici kliknemo na gumb za nastavitve, nato pa izberemo zavihek "Podatkovna mreža 2".

Prav tako lahko prilagoditev širine stolpcev izberemo v popup menuju tabele podatkov, in sicer pod izbiro 5.



Slika A 12: Nastavitve za širino stolpcev

Barve za posamezni tip podatkov lahko določimo tako, da v menuju Tools izberemo opcijo FieldTypeColors, ali pa v nastavitvah, in sicer na zavihku "Podatkovna mreža 3".



Slika A 13: Določanje barv glede na tip podatkov

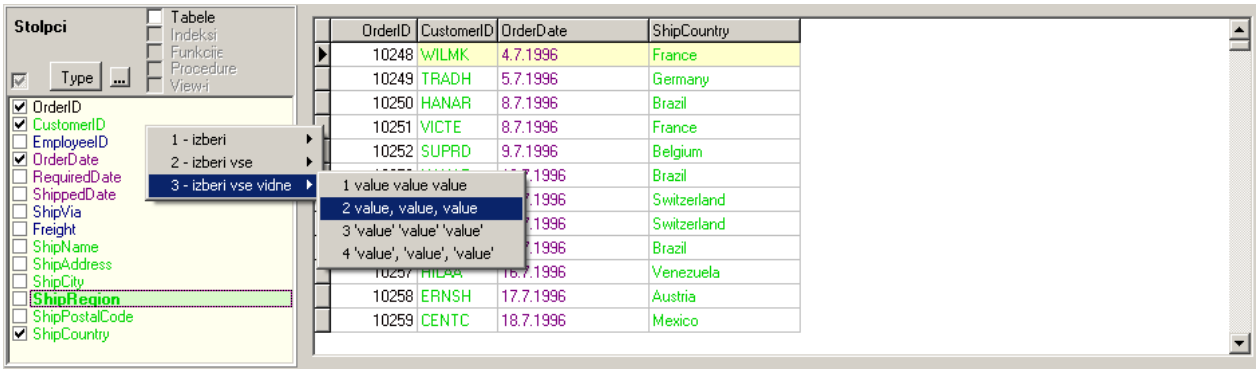
V tabeli rezultatov lahko spreminjamo vrstni red stolpcev, in sicer tako, da z levo tipko miške kliknemo na vrh stolpca in ga premaknemo na želeno mesto. Po premiku se ustrezno posodobi tudi seznam polj.

V seznamu polj pa imamo možnost, da stolpce, ki jih ne potrebujemo, skrijemo, in sicer tako, da odstranimo kljukico pred imenom polja.

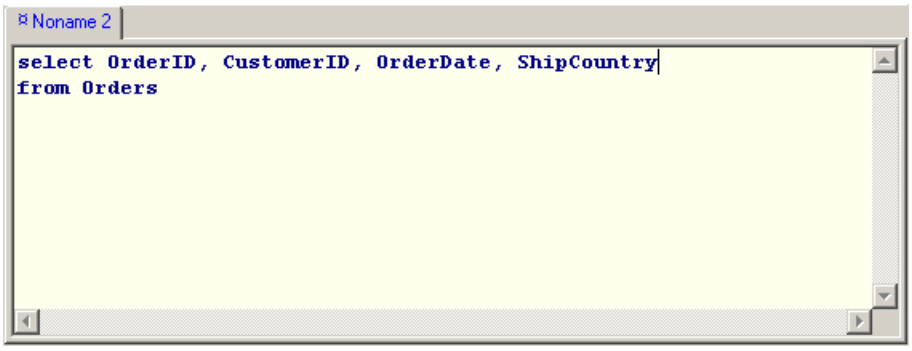
Če nad seznamom polj kliknemo na gumb "Type", se v seznamu namesto imen polj izpiše tip polja, v oglatih oklepajih pa še definirana velikost polja v bytih ter širina stolpca v pikslih.

Slika A 14: Seznam tipov in velikosti polj

V popup menuju pri seznamu polj imamo na voljo različne opcije za izbiro imen polj, in sicer lahko izberemo eno polje (trenutno izbrano), vsa polja ali pa samo tista, ki so vidna, se pravi obkljukana. Glede formata pa lahko izbiramo, ali so imena ločena samo s presledkom ali pa z vejico in presledkom ter še tem, ali se imena polj izpišejo v narekovajih ali ne.



Slika A 15: Izbira polj iz seznama



Slika A 16: Vpis izbranih polj v urejevalnik

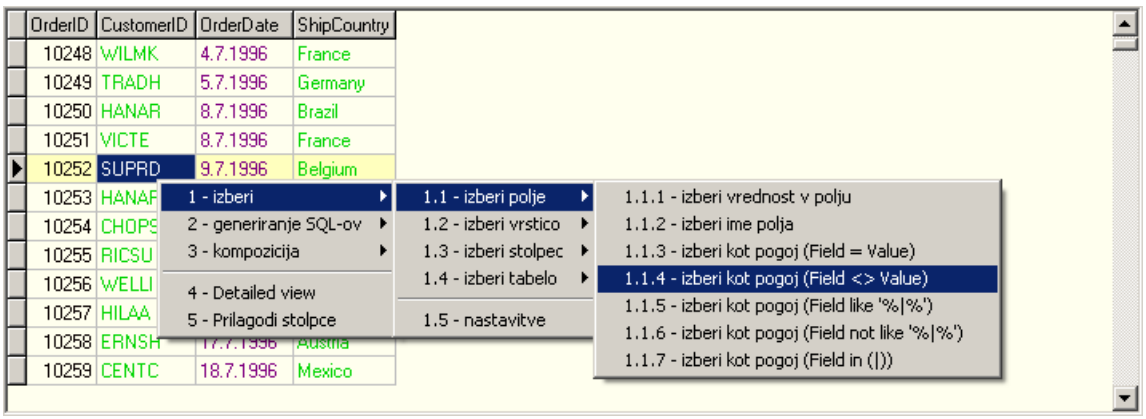
Prav tako se ime polja iz seznama vpiše v urejevalnik z dvojnim klikom na ime polja.

3.4.2 Izbira podatkov iz tabele podatkov

V popup menuju imamo pod opcijo 1 na voljo več možnosti izbiranja podatkov iz tabele.

Posamezne izbire se lahko nanašajo na posamezno polje, celotno vrstico, celoten stolpec ali pa kar celotno tabelo.

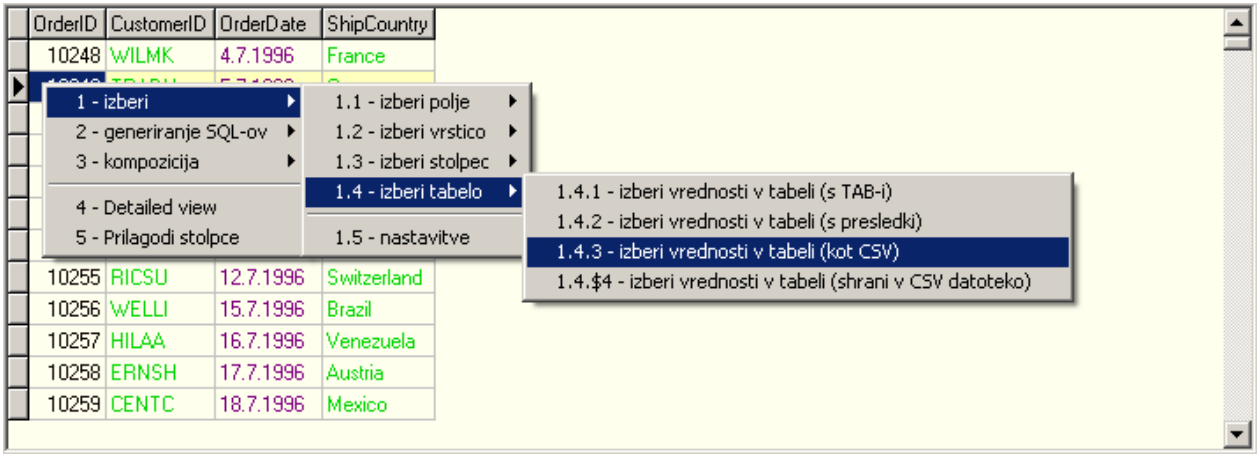
Pri izbiranju polja imamo nadalje možnosti, da izberemo vrednost v polju ali ime polja ali pa da generiramo del SQL stavka, ki predstavlja pogoj.



Slika A 17: Primer izbire polja iz tabele podatkov

Izbira vrstice in stolpca se redkeje uporablja in je pravzaprav ostanek razvoja.

Pri izbiri tabele pa imamo tudi več možnosti, in sicer vse podatke v tabeli lahko izberemo tako, da so posamezna polja ločena s tabulatorji ali presledki ali pa se tabela izpiše ali pa shrani v formatu CSV.

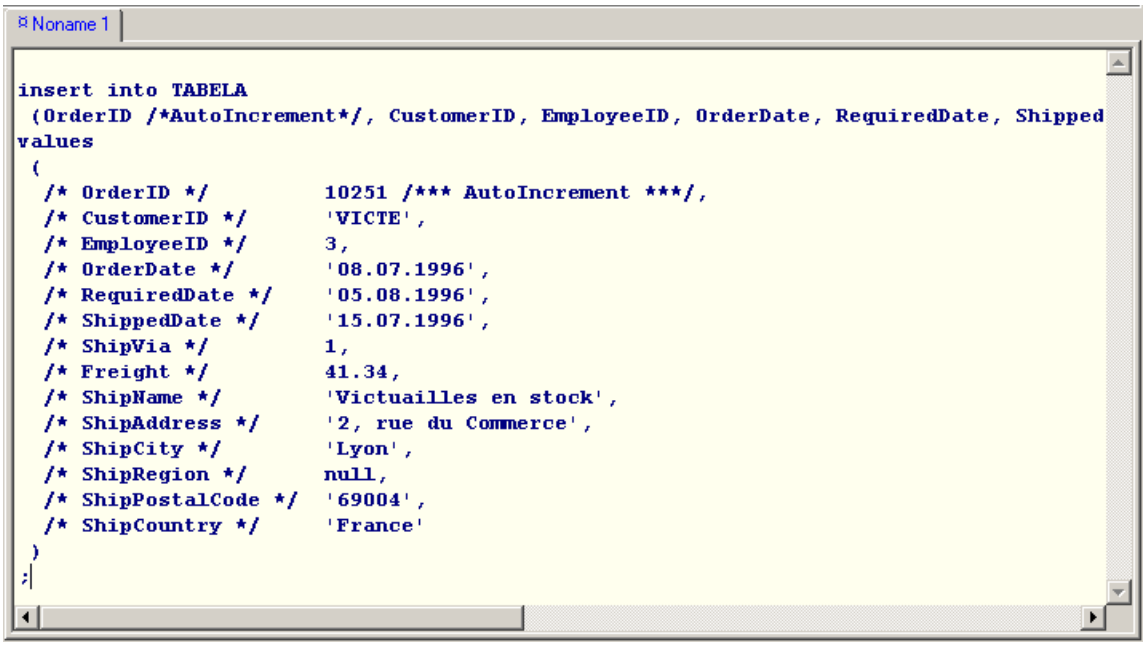


Slika A 18: Primer izbire tabele

3.4.3 Generiranje SQL stavkov

Iz podatkov v tabeli imamo možnost na hitro generirati INSERT stavek za eno vrstico ali za celotno tabelo ter DELETE in SELECT stavek.

Ti generirani stavki običajno predstavljajo osnovo za razvoj queryjev, saj je običajno veliko hitreje pobrisati nekaj odvečnih pogojev kot pa nekaj manjkajočih napisati.



Slika A 19: Primer generiranega INSERT stavka

3.4.4 Kompozicija

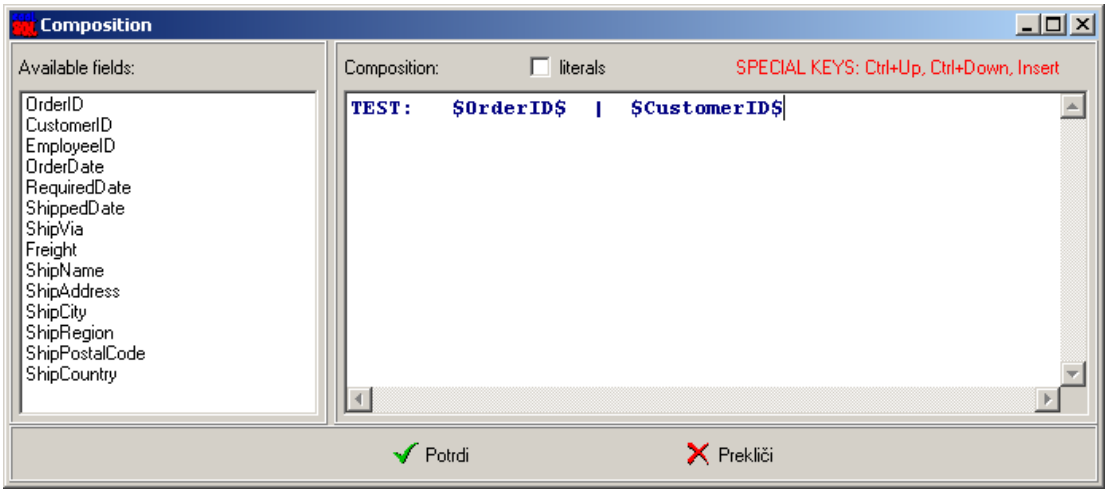
Včasih želimo podatke na hitro prikazati v nekem poljubnem formatu. Kompozicija nam omogoča definirati popolnoma poljuben format.

V popup menuju izberemo opcijo 3.

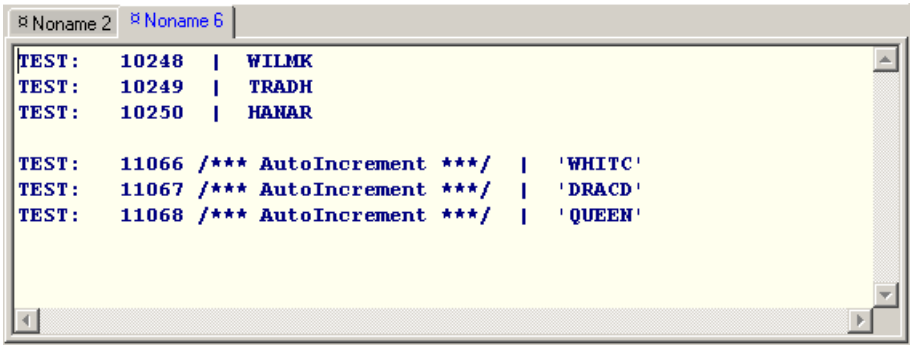
Kompozicijo moramo najprej definirati, in sicer to naredimo pod opcijo 3.3.

V urejevalnik lahko zapišemo besedilo, ki se bo izpisalo, na mesta, kjer pa se bodo vnesli podatki iz tabele, pa z dvojnim klikom na seznam polj vpišemo ustrezno polje.

Če nad urejevalnikom vključimo izbiro "literals", to vpliva na izpis podatkov iz tekstovnih ter še nekaterih drugih polj, npr. ftAutoIncrement.



Slika A 20: Primer definicije kompozicije

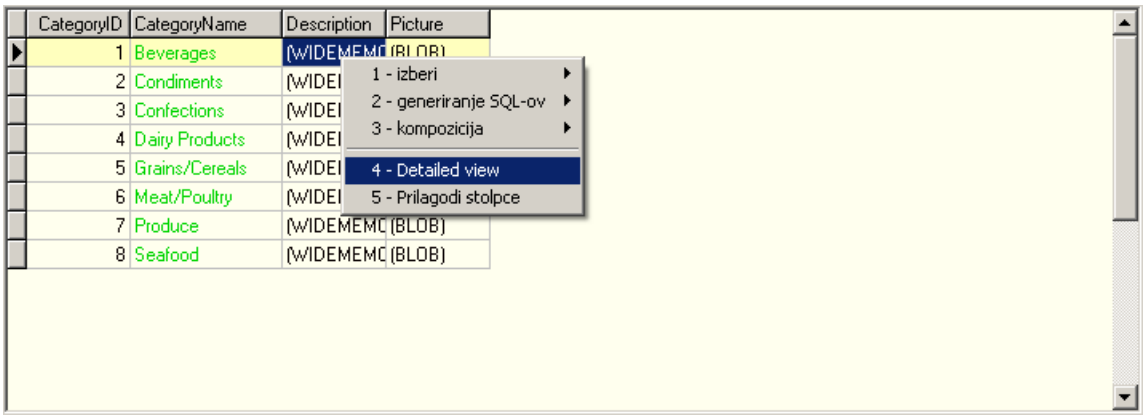


Slika A 21: Primer izbiranja s kompozicijo
Prve 3 vrstice so brez, druge tri pa z obkljukano izbiro "literals".

3.4.5 Podroben pogled

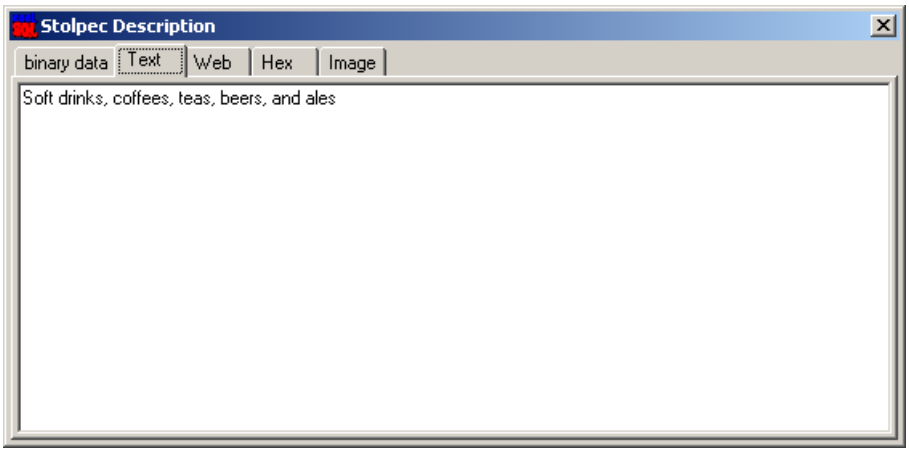
Včasih nekaterih podatkov v tabeli ni možno prikazati, npr. dolgih tekstovnih (npr. MEMO, WIDEMEMO) ali binarnih polj (BLOB).

Za rešitev tega problema je bil narejen podroben pregled. Odpremo ga v popup menuju pod opcijo 4.

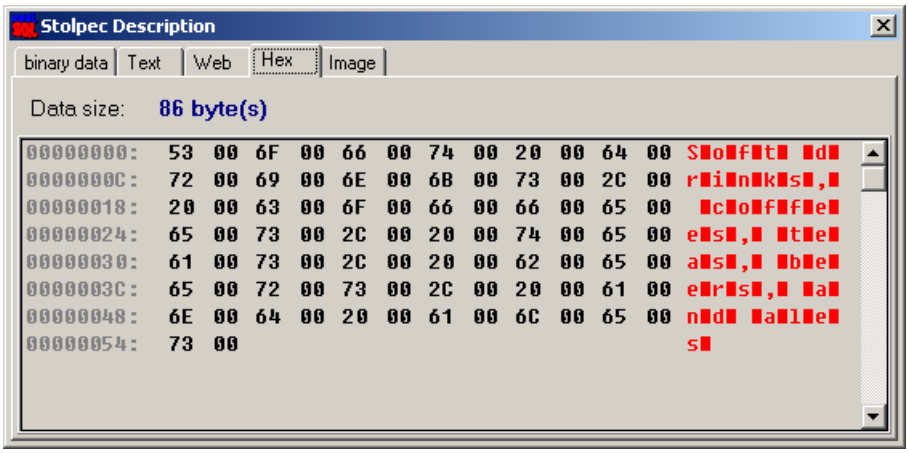


Slika A 22: Aktiviranje podrobnega pogleda

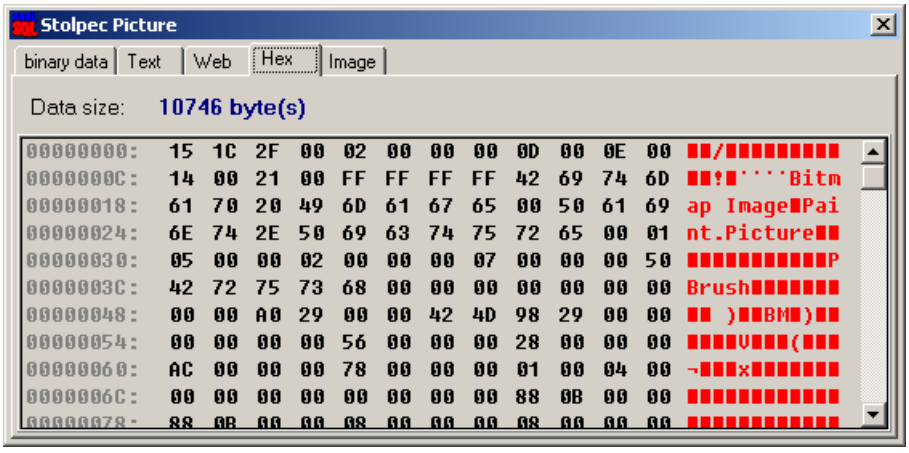
Podroben pogled nam omogoča, da isti podatek lahko gledamo na več načinov. Lahko ga prikažemo kot tekst ali pa ga v HEX editorju pogledamo na nivoju posameznih bytov. Opciji "Web" in "Image" sta bili uporabljani na nekem konkretnem projektu, kjer se je v BLOB polje shranjevala XML koda oz. slika v BMP formatu.



Slika A 23: Prikaz MEMO polja kot tekst



Slika A 24: Prikaz istega podatka v HEX oknu



Slika A 25: Prikaz BLOB polja v HEX oknu

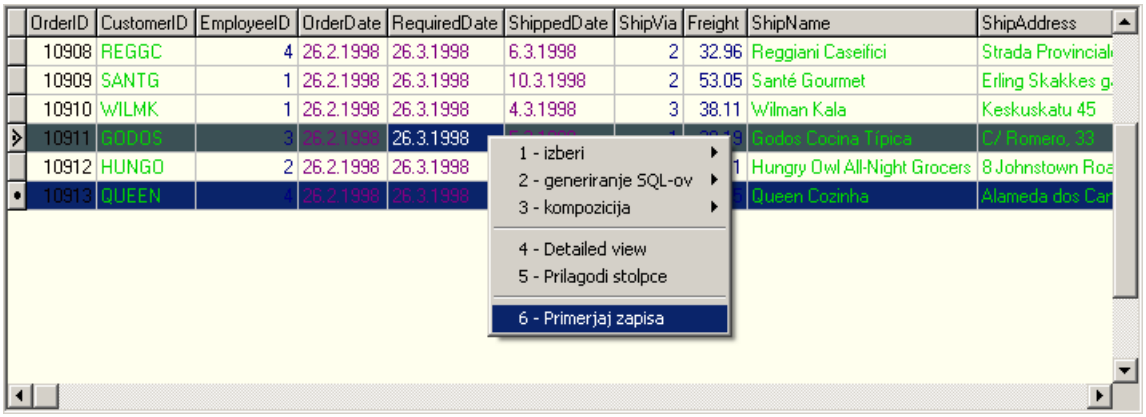
3.4.6 Primerjava zapisov

Ko razvijamo nek query, ponavadi že vemo, kakšen mora biti rezultat. Vendar se dostikrat zgodi, da query vrne preveč zapisov, pa ne moremo enostavno ugotoviti, kaj je narobe, kateri pogoj smo pozabili ipd. Sploh se to velikokrat zgodi, če povezujemo podatke iz več tabel. Kak zapis v rezultatu je običajno "podvojen", pa vendar ne bi smel biti.

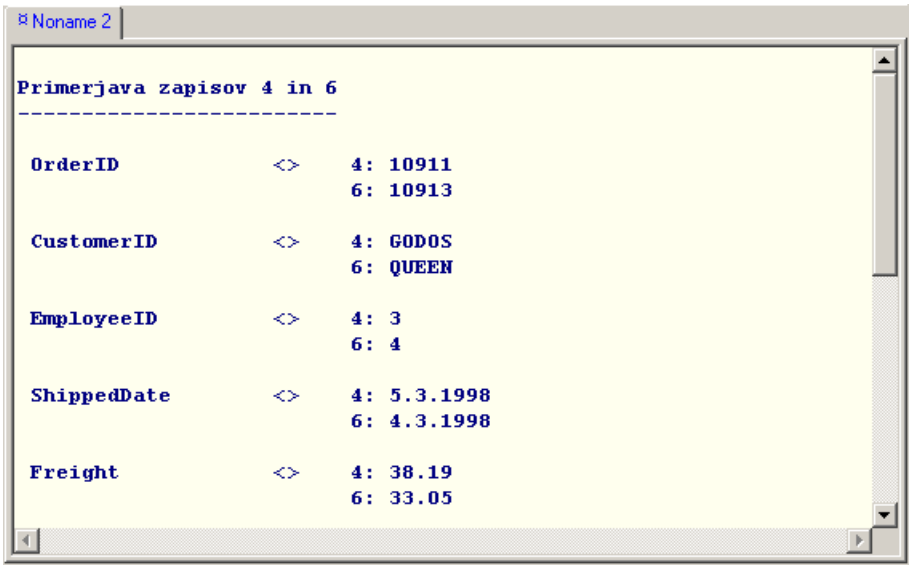
Če nam v tabeli uspe najti oba "podvojena" zapisa, ju lahko označimo tako, da držimo tipko Ctrl, z levo tipko miške pa označimo oba zapisa. Nato pa v popup menuju izberemo opcijo 6.

V urejevalniku se izpišejo samo tista polja, katerih vrednosti se pri izbranih zapisih razlikujejo.

Primerjamo lahko od 2 do 5 zapisov.



Slika A 26: Primerjava dveh zapisov



Slika A 27: Rezultat primerjave zapisov

3.5 Merjenje časa izvajanja SQL stavka

Najprej je bilo merjenje časa implementirano z Delphijevo funkcijo Now. Vendar je to omogočalo merjenje časa do le nekaj 10 milisekund natančno.

Z uporabo API QueryPerformanceCounterja pa se je natančnost merjenja časa izvajanja povečala za nekaj 100-krat.

Slika A 28: Primer trajanja izvajanja queryja (11,71 milisekund)
Čas izvajanja queryja je prikazan na desni strani statusne vrstice.

3.6 Ostala orodja

3.6.1 Iskanje vrednosti

V menuju "Tools" izberemo "Seek Value".

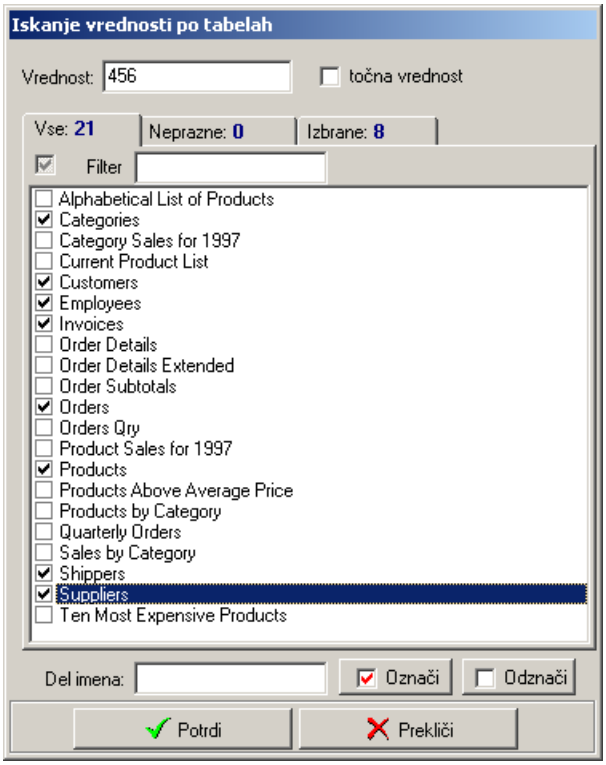
V okno vpišemo, katero vrednost iščemo. Lahko obkljukamo, če iščemo točno vrednost. V seznamu tabel izberemo tiste, po katerih bomo preiskovali.

Na voljo imamo tri sezname tabel na treh zavihkih, in sicer "Vse", "Neprazne" in "Izbrane". Tabele lahko izbiramo posamično – vsako posebej lahko vključimo (obkljukamo oz. označimo) ali izključimo (odkljukamo oz. odznačimo).

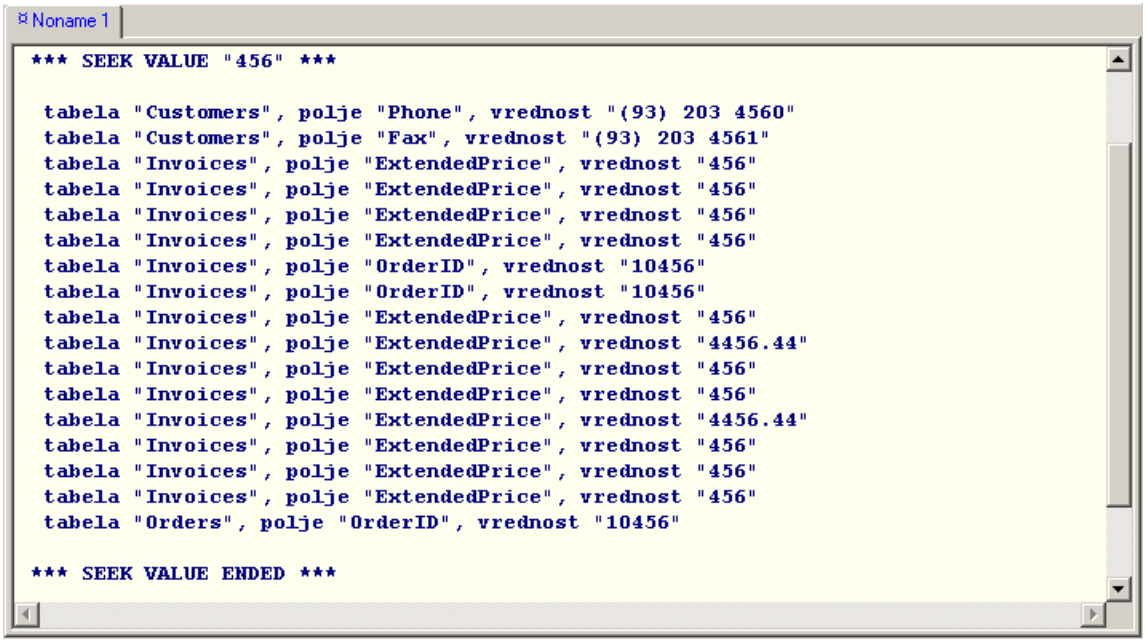
Nad seznamom tabel pa imamo tudi "skupni" checkbox, ki je lahko v treh stanjih:

- črna kljukica – v izbor so vključene vse tabele;
- brez kljukice – v izbor ni vključena nobena tabela;
- siva kljukica – v izbor so vključene tabele, ki smo jih izbrali posamično.

V izbor pa lahko vključimo ali pa iz njega izključimo skupino tabel, in sicer lahko v polje pod seznamom tabel vpišemo del imena, nato pa z gumbom "Označi" vključimo ali pa z gumbom "Odznači" izključimo tabele, katerih del imena se ujema z vpisanim "filtrom".



Slika A 29: Iskanje vrednosti po bazi



Slika A 30: Rezultat iskanja vrednosti

3.6.2 CSV izvoz

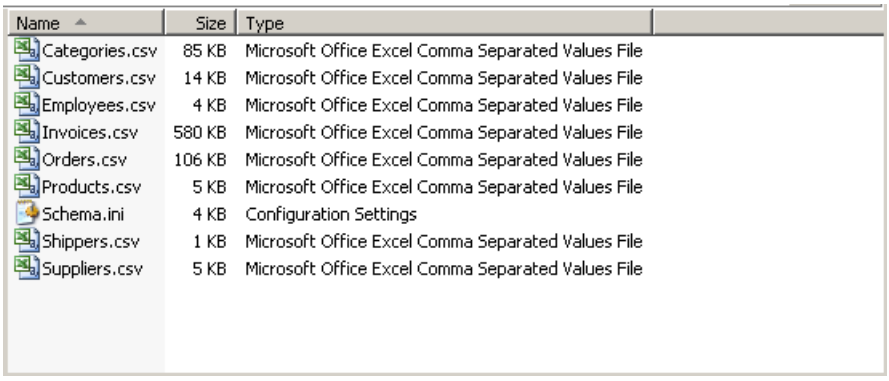
Program omogoča, da celotno bazo ali pa samo izbrane tabele izvozimo v CSV format. Lahko izvozimo samo strukturo baze, lahko samo podatke ali pa oboje.

CSV izvoz prikličemo s pritiskom na gumb "CSV" v orodni vrstici.

Tabele izberemo na enak način kot pri "Iskanju vrednosti" (točka 2.6.1).

Določiti moramo tudi pot (mapo, direktorij), kamor se baza izvozi.

Struktura baze se izvozi v datoteko Schema.ini, vsaka od izbranih tabel pa v svojo CSV datoteko.



Slika A 31: Primer izvoza izbranih tabel

CSV izvoz je uporaben predvsem zato, ker shrani tako strukturo baze kot podatke v nekem trenutku. Če npr. naredimo izvoz pred in po tem, ko testiramo neko aplikacijo, ki "nekaj počne" z

bazo, lahko s primerjavo obeh izvozov ugotovimo razlike, ki jih je "naredila" testirana aplikacija. Za primerjavo map in vsebine samih datotek obstaja kar nekaj programov, npr. Beyond Compare, Compare It!, WinMerge, Meld, ExamDiff Pro, Diffuse ...

4. Zaključek

Iz potrebe po preprostem orodju za preverjanje pravilnosti SQL stavkov se je coolSQL razvil v kar obsežen projekt. Če bi moral napisati aplikacijo na novo, bi k temu projektu pristopil sistematičneje, saj sedaj vem, kaj vse naj bi aplikacija omogočala.

Kar se tiče nadaljnega razvoja coolSQL-a, se bodo nove funkcionalnosti še naprej dodajale po potrebi.

Mogoče se bom kdaj lotil tudi tega, da bom delovanje obstoječih funkcionalnosti razširil še na druge bazo, ne samo tisto, za katero sem posamezno funkcionalnost potreboval.

V kolikor mi bo čas dopuščal, bom nadgradil urejevalnik tako, da bo podpiral sintaktično barvanje ključnih besed, številk, nizov, komentarjev ... (t. i. syntax highlighting).

Verjetno bom program tudi preimenoval, saj sem pred nedavnim izvedel, da že obstaja aplikacija z imenom "CoolSQL" (http://coolsql.sourceforge.net/).

5. Viri

[1] Standardna pomoč v Delphi 6

[2] Match column title alignments with field alignments in DBGrid
http://delphi.about.com/od/adptips2003/a/bltip1103_2.htm

[3] Coloring the TDBGrid Delphi component
http://delphi.about.com/od/usedbvcl/l/aa031699.htm

[4] TreeView Items - Dynamic Context (Popup) Menus in Delphi
http://delphi.about.com/od/delphitips2008/qt/treeitem_popup.htm

[5] Format Date_Time Values for Access SQL in Delphi
http://delphi.about.com/od/delphitips2007/qt/datetime_sql.htm

[6] Textfile connection strings - ConnectionStrings.com
http://www.connectionstrings.com/textfile/

[7] Schema.ini File (Text File Driver)
https://msdn.microsoft.com/en-us/library/ms709353.aspx

[8] High Performance Timer in Delphi – TstopWatch
http://delphi.about.com/od/windowsshellapi/a/delphi-high-performance-timer-tstopwatch.htm

[9] SwissDelphiCenter.ch _ ...autosize a dbgrid-column to fit its contents_
http://www.swissdelphicenter.ch/en/showcode.php?id=2041

[10] How to Refresh a DBGrid without Losing the Current Row Position
http://delphi.about.com/od/delphitips2008/qt/dbgrid_row_pos.htm

[11] error '80040e07' wrong syntax in Date expression - ASP _ Active Server Pages
https://bytes.com/topic/asp-classic/answers/127232-error-80040e07-wrong-syntax-date-expression